



# Línguagem de Programação Java - Avançado

---

# Sumário

Apresentação	4
<b>Módulo 1</b>	<b>6</b>
Fundamentos da programação orientada a objetos (POO)	6
Fundamentos da Programação Orientada a Objetos (POO)	7
<i>Integrated Development Environment (IDE)</i>	7
Instalando a IDE Eclipse	9
Objetos, classes e atributos	11
Construindo classes	16
Construindo objetos	17
<b>Módulo 2</b>	<b>20</b>
Complementos da linguagem de programação Java	20
Complementos da linguagem de programação Java	21
Pacotes ( <i>package</i> )	21
Método com parâmetros	25
Método com retorno	33
Métodos <i>get()</i> e <i>set()</i> (encapsulamento)	36
Herança	38

# Sumário

Módulo 3	47
Tratamento de exceções, interface gráfica, eventos e integração	47
Tratamento de exceções, interface gráfica, eventos e integração	48
Tratamento de exceções	48
Exception	49
Finally	52
Interface gráfica	55
GUI layout, border e flow layout	60
Eventos	64
Integração com banco de dados <i>Oracle Express Edition</i>	66
Instalando o <i>Oracle Express Edition</i>	67
Utilizando o driver JDBC para conectar o banco de dados	75
Fechamento	81
Referências	82

# Apresentação

Olá!

Bem-vindo(a) ao curso **Linguagem de Programação Java - Avançado!**

O objetivo deste curso é apresentar os principais conceitos relacionados à linguagem de programação Java, por isso, serão evidenciados os fundamentos da linguagem de Programação Orientada a Objetos (POO), passando pelos complementos da linguagem, como pacotes, métodos e herança. Além disso, você entenderá como realizar o tratamento de exceções, de que modo funciona a interface gráfica por meio do IDE Eclipse e terá uma breve visão dos eventos. Por fim, você fará um exercício guiado por um passo a passo, incluindo a instalação de um banco de dados e como conectar o programa Java utilizando o driver JDBC.

Antes de iniciar seus estudos, assista ao vídeo a seguir para uma introdução quanto ao que será visto ao longo do curso.

Desejamos a você um excelente aprendizado!



## Vídeo

Confira o [vídeo](#) de apresentação do curso.

Perdeu algum detalhe? Confira o que foi abordado no vídeo.

Olá! Seja bem-vindo ao curso de Linguagem de Programação Java - Avançado.

Aqui você verá os fundamentos da Programação Orientada a Objetos, também conhecida como POO. Além de poder estudar sobre as ferramentas de desenvolvimento mais completas e como instalá-las para serem utilizadas no dia a dia profissional e, até mesmo, pessoal.

Também conhecerá os objetos, as classes e suas construções, além dos complementos da linguagem de programação Java, incluindo conceitos como pacotes, métodos e herança.

Por fim, você estudará de modo mais aprofundado o tratamento de exceções, interface gráfica, eventos e integração.

Com atenção e colocando os conhecimentos em prática, este conteúdo será muito valioso para você crescer ainda mais em sua área de atuação, tornando-se um verdadeiro expert nessas temáticas.

Vamos começar nossa jornada?



Módulo 1

# Fundamentos da Programação Orientada a Objetos (POO)

---

# Fundamentos da Programação Orientada a Objetos (POO)

Neste módulo, você aprenderá o que é uma IDE, por qual motivo devemos utilizá-la e como instalar o IDE Eclipse. Estudará, ainda, sobre os conceitos de objetos, classes e atributos. Vamos iniciar?

## ***Integrated Development Environment (IDE)***

No mundo corporativo, os desenvolvedores, em seu cotidiano, costumam utilizar ferramentas de desenvolvimento mais completas, conhecidas como *Integrated Development Environment* (IDE). Em tradução livre, temos algo como “ambiente de desenvolvimento integrado”. Trata-se de uma ferramenta que possui todos os recursos necessários em um só lugar.

Para conhecer um pouco mais a respeito das IDEs, ouça o *podcast* na sequência!



### ***Podcast***

Confira o [\*podcast\*](#) sobre as IDEs.

Perdeu algum detalhe? Confira o que foi abordado no *podcast*.

Olá! Vamos falar sobre a *Integrated Development Environment* ou as IDEs?

As IDEs são utilizadas para aumentar a produtividade, para que o desenvolvedor possa entregar os softwares do seu projeto de trabalho com qualidade. Isso, porque as IDEs auxiliam, principalmente, na qualidade dos códigos que serão construídos. Confira, agora, quais são seus principais recursos.

Dentro de uma IDE, temos um editor de códigos, o qual compila e executa os programas em Java, além de disponibilizar recursos que complementam os comandos quando iniciamos a digitação. Esse editor também auxilia no processo de *debug* do código, trazendo a possibilidade de identificar e localizar os erros.

Outras vantagens da ferramenta é que conseguimos controlar o versionamento dos programas desenvolvidos em Java. Ou seja, entender o que foi elaborado em determinado período, sendo possível, quando necessário, voltar ao código anterior que estava funcionando.

Ela ainda permite o acesso a documentação da linguagem Java, a fim de identificar quais são os métodos e atributos das classes do pacote Java Lang. Sendo assim, podemos perceber que a IDE nada mais é que um software utilizado para construir outro software, isto é, trata-se também de um programa.

As principais IDEs do mercado ou mais populares - uma vez que muitas pessoas as utilizam no cotidiano - são as IDEs Eclipse e NetBeans. Mas também existem outras comumente utilizadas, como a IntelliJ, JBuilder, JCreator, IBM RAD, BlueJ, Jdeveloper, JGrasp e Kdeveloper.

Com esse *podcast*, você pôde perceber que as IDEs são recursos essenciais para otimizar a criação de softwares, fazendo com que você possa criar linhas de programação com qualidade, diminuindo os erros e as inconsistências, bem como gerir adequadamente esse processo sem perder de vista a demanda dos clientes.

Viu só as vantagens em utilizar as IDEs para o aumento da produtividade e demais processos que ela permite? Que tal instalá-las para ajudar em seu trabalho?

Resumidamente, entre as principais vantagens de se utilizar uma IDE, podemos mencionar: aumento da produtividade, edição de códigos, compilação, JVM, *autocomplete*, DEBUG, controlar versões, documentação e correção de erros.

## Instalando a IDE Eclipse

Agora que você compreendeu o que é uma IDE, quais seus tipos mais conhecidos e as vantagens de utilizarmos essa ferramenta, chegou o momento de instalar a IDE Eclipse para realizar a construção dos seus programas.

Para iniciar a instalação, acesse o [site para download](#). Depois, acompanhe o passo a passo no vídeo a seguir!



### Vídeo

Confira o [vídeo](#) de instalação da IDE Eclipse.

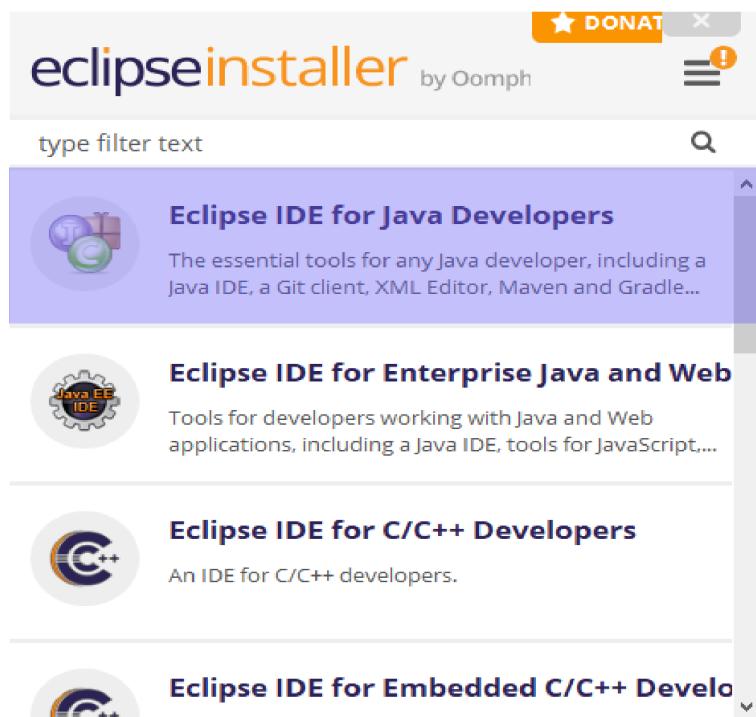
Perdeu algum detalhe? Confira o que foi abordado no vídeo.

Olá! Agora que você já sabe o que é uma IDE, vamos instalar a IDE Eclipse? Para iniciar a instalação, acesse o [site](#) e faça o download necessário.

Em seguida, aparecerá uma página para você clicar em “Download” novamente.

Após a finalização do download, acesse onde você salvou o arquivo e dê dois cliques para começar.

Logo antes de iniciar a instalação, abrirá esta tela, onde você deverá selecionar uma das opções dela, conforme podemos observar: “*Eclipse IDE for Java Developers*”.



Feita a seleção, indique o melhor local para instalar e clique no botão “*Install*” e, depois, em “*Accept Now*”. Feito isso, temos a indicação de que a instalação se iniciará.

Ao finalizar, aparecerá a opção “*Launch*” para abrir a IDE Eclipse. Feito isso, você precisará selecionar um diretório local para poder ser o espaço de trabalho e dar um “*Launch*” novamente para sair.

Em seguida, assim que acessar a Eclipse, aparecerá a tela inicial, a qual deverá ser fechada para visualizar a tela de programação.

Com este passo a passo detalhado, fica mais fácil, não é mesmo?

Agora é só aguardar a instalação terminar.

Como você viu no vídeo, é bem simples realizar a instalação e a configuração da ferramenta IDE Eclipse. Com essa etapa finalizada podemos prosseguir com os estudos. A seguir, você conhecerá os conceitos de objetos, classes e atributos. Vamos lá!

## Objetos, classes e atributos

Java é uma linguagem de programação orientada a objetos, criada com a ideia fundamental de aproximar o mundo real do virtual (computador). Para tanto, há a necessidade do uso de **objetos**, pois o mundo é composto por eles. Vamos pensar em um exemplo?



Imagine o objeto carro. Para você dirigí-lo, precisa que este item seja projetado e construído. Somente depois poderá entrar, ligar e acelerar o carro, não é mesmo? Isso também ocorre em relação a qualquer outro objeto. Assim, este, para ser utilizado dentro do programa Java, precisa também ser projetado. Com base no projeto, construímos o objeto!

### #PraCegoVer

Na imagem, há um carro esportivo de cor vermelha com um condutor dentro dele. O veículo está em alta velocidade e em uma estrada com uma floresta.

O objeto cachorro, por exemplo, poderia ser representado pela classe Cachorro. Esta **classe** é o projeto do objeto e apresenta o tipo que se pretende construir. Além disso, temos que os objetos são compostos de atributos e ações. Os **atributos** são representados pelas variáveis, enquanto as ações são representadas por métodos.

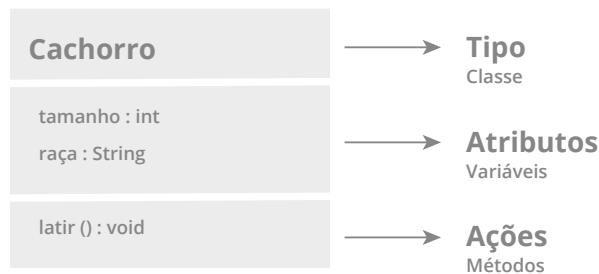


#### #PraCegoVer

Na fotografia, há um cachorro de coleira sentado no meio de uma sala com uma parte de um sofá ao fundo. O rosto dele está olhando para a câmera, enquanto o que está ao fundo tem uma imagem desfocada.

Desse modo, todo objeto dentro do Java apresenta determinado tipo e pode conter atributos e ações. No entanto, para que você realmente entenda, vamos analisar esses conceitos de maneira mais aprofundada?

Um objeto cachorro pode ter diversos atributos, como as orelhas, os olhos e a coleira, assim como pode ser de determinada raça e ter um tamanho específico. Com isso, independentemente da espécie de cachorro, todo objeto criado com essas particularidades apresentará esses tipos de atributos, mas com valores diferentes, ou seja, com uma orelha maior ou menor, tamanho diferente, e assim por diante.



**#PraCegoVer:** na imagem, temos um cachorro sentado representando o objeto cachorro. Do lado esquerdo, encontramos uma ilustração do animal. Do lado direito, temos o tipo (classe), os atributos (variáveis) e as ações (métodos).

E quanto às ações que o cachorro pode executar?

Pois bem, ele pode fazer xixi, abanar o rabo, enterrar ossos, correr, comer, latir e dormir!

A classe, como já sabemos, não é um objeto. Ela faz parte do projeto de um objeto, sendo que os objetos que criamos a partir da classe são chamados de **instâncias de objeto**.

Como você pode conferir a partir de uma classe Cachorro, podemos criar vários cachorros, cada um com propriedades diferentes.

Para que esse conselho fique claro, acompanhe mais um exemplo, observe abaixo.

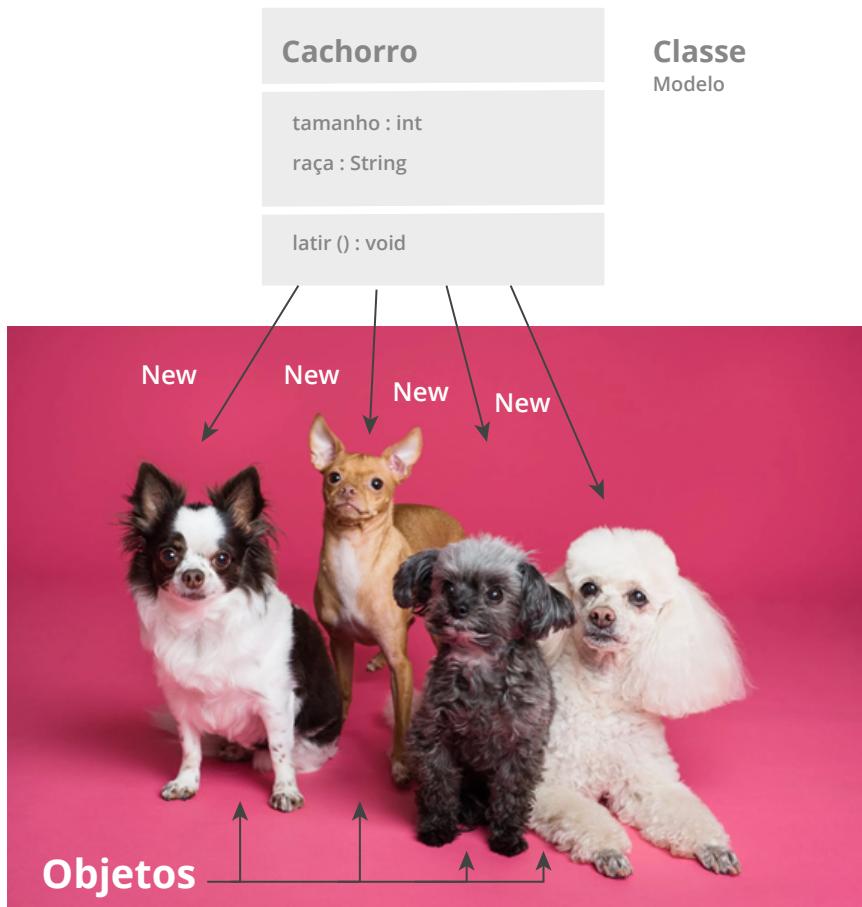
### Situação 1

Por exemplo, imagine um bloco de cartões contendo endereços. Nesses blocos, que devem estar em branco, encontramos três propriedades: nome, telefone e e-mail. Quando escolhemos um dos cartões e o preenchemos com as informações solicitadas, o cartão deixa de ser um modelo e se torna um objeto, pois contém os dados e atributos.

## Situação 2

Agora, se o cartão fosse representado dentro de um programa Java, ele poderia conter alguns métodos, além dos atributos, como nome, telefone e e-mail.

Dessa maneira, considerando a segunda situação, teríamos um método chamado **consultar nome**. Observe abaixo o conceito retratado.



**#PraCegoVer:** na imagem, temos quatro cachorros de raças diferentes, representando a classe (dos cachorros e suas características principais) e seus objetos (cada cachorro de uma espécie diferente). Na parte superior, temos a classe (modelo), a qual é dividida em quatro raças diferentes de cachorros.

Seguindo essa lógica, todos os objetos teriam condições de consultar nome, porém, cada um conhece apenas o próprio nome, telefone e e-mail.



**#PraCegoVer:** na imagem, existe uma caixa com várias pastas coloridas e uma mão tirando um papel de uma dessas divisões internas. Neste papel está escrito a palavra "Design".

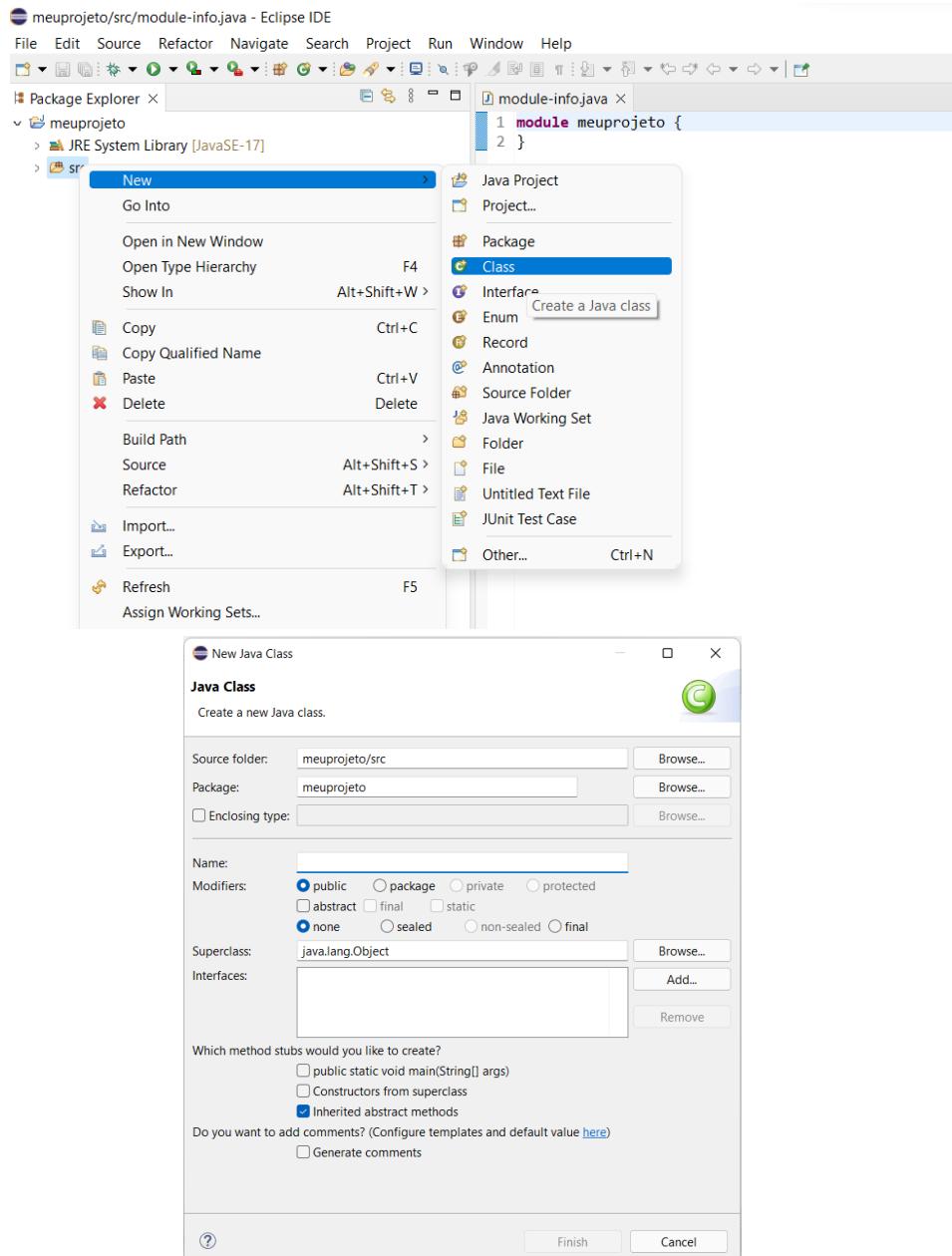
Conforme você estudou, a linguagem Java aproxima o mundo real do virtual a partir da relação de objetos (seres e coisas presentes no mundo real), de classes (nome dado ao grupo de objetos com características comuns) e atributos (as características comuns e específicas de cada objeto pertencente à classe).

Por isso é importante dominar esses conceitos, pois, no momento de você aplicar a linguagem Java na programação, ela deve se organizar dessa forma.

Assim, no próximo tópico, você aprenderá a como construir classes. Confira!

## Construindo classes

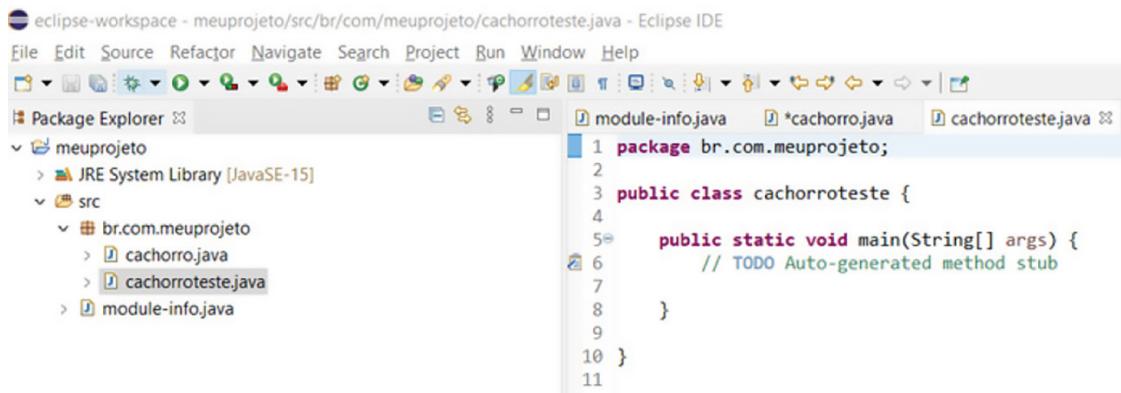
Iniciaremos esse tópico construindo duas classes, uma sendo a implementação do projeto cachorro e outra que irá chamá-lo para acessar cada uma das suas propriedades, seus atributos e seu método.



**#PraCegoVer:** na imagem temos um print da tela do software Eclipse com dashboard, barra de ferramenta na parte superior, com o layout de pastas no canto esquerdo e com o painel de programação no lado direito. No centro, há uma janela aberta a partir do acionamento do botão direito do mouse indicando como ativar a opção "class". Na imagem da direita, existe outro print da tela do software Eclipse referente à janela de criação e edição da opção "class".

Inicialmente, você precisa abrir a IDE Eclipse. Selecione com o botão direito do mouse em “src”, optando por “New” e, na sequência, selecione a opção “Class”.

Na tela seguinte, no campo “Name”, digite “cachorro”.



The screenshot shows the Eclipse IDE interface. The top menu bar includes File, Edit, Source, Refactor, Navigate, Search, Project, Run, Window, and Help. Below the menu is a toolbar with various icons. On the left, the Package Explorer shows a project named 'meuprojeto' containing a 'src' folder which has a 'br.com.meuprojeto' package with files 'cachorro.java' and 'cachorroteste.java', and a 'module-info.java' file. The right side of the interface contains three tabs: 'module-info.java', '\*cachorro.java', and 'cachorroteste.java'. The 'cachorroteste.java' tab is active, displaying the following Java code:

```
1 package br.com.meuprojeto;
2
3 public class cachorroteste {
4
5     public static void main(String[] args) {
6         // TODO Auto-generated method stub
7
8     }
9
10 }
11
```

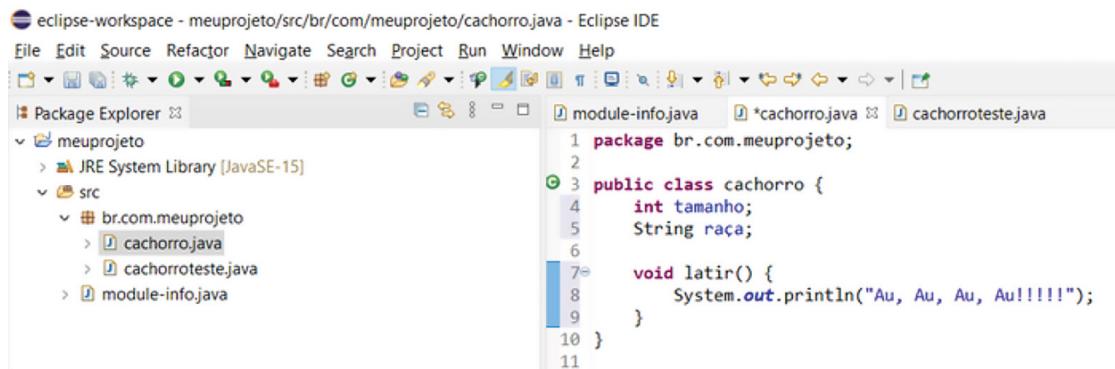
**#PraCegoVer:** na imagem, temos um *print* da tela do software Eclipse com *dashboard*, barra de ferramenta na parte superior, com o layout de pastas no canto esquerdo e com o painel de programação no lado direito, em que está escrito o seguinte código:  
package br.com.meuprojeto; public class cachorroteste { public static void main (String[] args) { // TODO Auto-generated method stub } }

Em seguida, realize o mesmo procedimento, mas agora para criar uma classe, sendo esta chamada de “cachorroteste”. Selecione a opção “public static void main (String[] args)”. Dessa forma, ficará com duas guias para as classes criadas, “cachorro.java” e “cachorroteste.java”.

Com essas duas classes criadas, temos parte do processo iniciado. No entanto, você saberia responder qual é o próximo passo? Se você pensou nos objetos, está no caminho certo. Mas como construir os objetos? É o que você verá a seguir!

## Construindo objetos

Com as guias que você criou anteriormente, podemos passar para a construção dos objetos das suas classes. Iniciaremos com a elaboração do objeto “cachorro.java”, conforme você pode observar na imagem.



The screenshot shows the Eclipse IDE interface. The top menu bar includes File, Edit, Source, Refactor, Navigate, Search, Project, Run, Window, and Help. Below the menu is a toolbar with various icons. On the left, the Package Explorer shows a project named 'meuprojeto' with a 'src' folder containing 'br.com.meuprojeto' and files 'cachorro.java' and 'cachorroteste.java'. A 'module-info.java' file is also listed. The right side of the interface displays the code for 'cachorro.java':

```

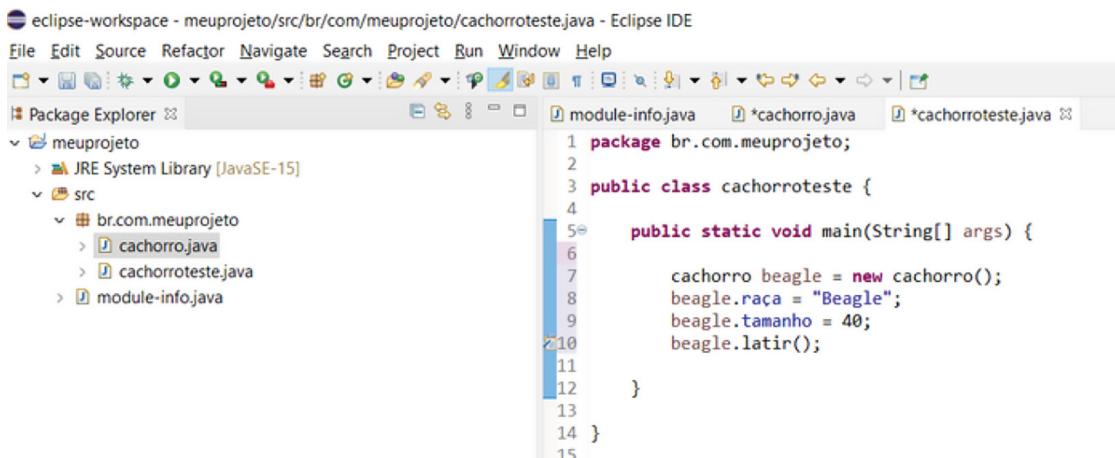
1 package br.com.meuprojeto;
2
3 public class cachorro {
4     int tamanho;
5     String raça;
6
7     void latir() {
8         System.out.println("Au, Au, Au, Au!!!!!");
9     }
10 }
11

```

**#PraCegoVer:** na imagem, temos um *print* da tela do software Eclipse com *dashboard*, barra de ferramenta na parte superior, com o layout de pastas no canto esquerdo e com o painel de programação no lado direito, em que está escrito o seguinte código:

```
package br.com.meuprojeto; public class cachorro { int tamanho; String
raça; void latir() { System.out.println("Au, Au, Au, au!!!!"); }}
```

Em seguida, vamos criar os objetos da sua classe “cachorroteste.java”. Confira na próxima imagem para entender o processo de forma mais detalhada.



The screenshot shows the Eclipse IDE interface. The top menu bar includes File, Edit, Source, Refactor, Navigate, Search, Project, Run, Window, and Help. Below the menu is a toolbar with various icons. On the left, the Package Explorer shows a project named 'meuprojeto' with a 'src' folder containing 'br.com.meuprojeto' and files 'cachorro.java' and 'cachorroteste.java'. A 'module-info.java' file is also listed. The right side of the interface displays the code for 'cachorroteste.java':

```

1 package br.com.meuprojeto;
2
3 public class cachorroteste {
4
5     public static void main(String[] args) {
6
7         cachorro beagle = new cachorro();
8         beagle.raça = "Beagle";
9         beagle.tamanho = 40;
10        beagle.latir();
11    }
12
13
14 }
15

```

**#PraCegoVer:** na imagem, temos um *print* da tela do software Eclipse com *dashboard*, barra de ferramenta na parte superior, com o layout de pastas no canto esquerdo e com o painel de programação no lado direito, em que está escrito o seguinte código:

```
package br.com.meuprojeto; public class cachorroteste { public static
void main(String[] args) { cachorro beagle = new cachorro(); beagle.raça =
"Beagle"; beagle.tamanho = 40; beagle.latir(); } }
```

Por fim, você precisa testar as classes selecionando o botão “Run”, presente no menu superior da IDE Eclipse. Observe o resultado abaixo:

The screenshot shows the Eclipse IDE interface. The top menu bar includes File, Edit, Source, Refactor, Navigate, Search, Project, Run, Window, and Help. The left sidebar is the Package Explorer, showing a project named 'meuprojeto' containing a 'src' folder with files 'cachorro.java', 'cachorroteste.java', and 'module-info.java'. The right side is the code editor with the following Java code:

```

1 package br.com.meuprojeto;
2
3 public class cachorroteste {
4
5     public static void main(String[] args) {
6
7         cachorro beagle = new cachorro();
8         beagle.raça = "Beagle";
9         beagle.tamanho = 40;
10        beagle.latir();
11    }
12
13 }
14
15
16
17

```

Below the code editor is the Console view, which displays the output: <terminated> cachorroteste [Java Application] C:\Users\cmoreno\p2\pool\plugins\org.eclipse.jdt.openjdk.hotsp Au, Au, Au, Au!!!!!

**#PraCegoVer:** na imagem, temos um *print* da tela do software Eclipse com *dashboard*, barra de ferramenta na parte superior, com o layout de pastas no canto esquerdo e com o painel de programação no lado direito, em que está escrito o seguinte código: package br.com.meuprojeto; public class cachorroteste { public static void main(String[] args) { cachorro beagle = new cachorro(); beagle.raça = "Beagle"; beagle.tamanho = 40; beagle.latir(); } } Na aba inferior denominada "Console", está escrito "au, au, au, au!!!!"

Parabéns! Você chegou ao final do primeiro módulo. Aqui você foi apresentado ao IDE, *Integrated Development Environment* (em tradução livre, “ambiente de desenvolvimento integrado”), que é um software que produz outros softwares por meio da linguagem Java.

Além disso, refletimos que, para usar tal recurso, é preciso dominar alguns conceitos específicos da linguagem Java, os quais vimos anteriormente. São eles: os objetos (seres e coisas presentes no mundo real os quais são representados pelos códigos de programação), as classes (nome dado ao grupo de objetos com características comuns representado pelos códigos de programação) e os atributos (as características comuns e específicas de cada objeto pertencente à classe).

Por fim, você pôde ver o tutorial de como instalar esse software, bem como programar tais elementos a partir de exemplos ilustrados do IDE. Desta forma, essa ferramenta e os conceitos relacionados a ela são essenciais para você, futuro desenvolvedor de softwares, uma vez que eles otimizam e criam programas com códigos adequados, proporcionando uma entrega de qualidade para o cliente.

Por conta da relevância desse programa, da linguagem e dos conceitos correlatos utilizados nele, no próximo módulo, vamos aprofundar os estudos sobre a linguagem Java, principalmente, a partir dos seus complementos, ou seja, os pacotes Java Lang e Java Util.



Módulo 2

# Complementos da linguagem de programação Java

---

# Complementos da linguagem de programação Java

Você aprendeu que o IDE, é um excelente programa voltado para criação de softwares, o qual usa a linguagem Java, bem como os conceitos de objetos, classes e de atributos.

Devido à sua importância, sobretudo porque é uma ferramenta bastante útil no contexto da programação orientada a objetos, é importante que você aprofunde o seu conhecimento sobre a linguagem Java.

Por conta disso, neste módulo, você estudará os pacotes Java Lang e Java Util, entendendo como aplicá-los nas linhas de programação. Verá também alguns métodos, quais os seus tipos, bem como o conceito de herança.

Sabendo disso, inicie essa nova etapa do curso!

## Pacotes (**package**)

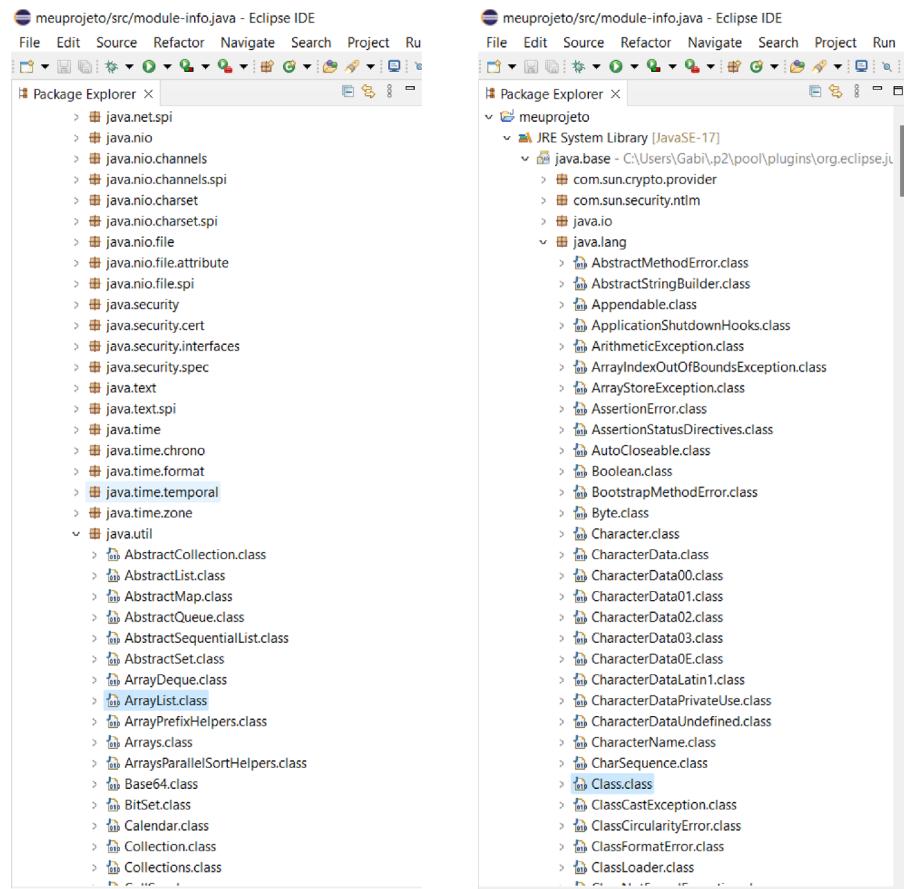
Iniciaremos esse tópico com três perguntas, e esperamos que ao final você tenha compreendido todos os temas que serão apresentados e saiba responder a todas elas:

O que é um pacote?

Por que você deve utilizar pacotes?

Como pode construí-los e utilizá-los dentro dos seus programas?

No pacote Java, há o “`java.lang`”, que é o pacote da linguagem de programação Java. Nele, você encontra todas as classes, sem precisar importar de outros pacotes. Por exemplo, você pode ter no pacote “`java.lang`” a classe `String.class`, ao passo que, no pacote “`java.util`”, terá a classe `ArrayList.class`.



**#PraCegoVer:** na imagem, temos um *print* da tela do *software* Eclipse. Do lado esquerdo, encontramos os pacotes de Java Lang. Do lado direito, temos os pacotes de Java Util.

Considerando isso, você observou, na imagem acima, que tanto o pacote “java.lang”, quanto o “java.util” têm o objetivo de agrupar elementos que possuem características em comum? Inclusive, isso ajuda muito na hora de programar, não é?

Isso mesmo! Pois essa é a função do conceito de pacote. Um pacote considera a estrutura de diretórios em que podemos colocar todas as classes, separando-as por determinado assunto. Portanto, trata-se de um agrupamento de classes organizadas por categoria, ou seja, uma forma de modularizar o programa. Para ficar mais claro, veja o exemplo a seguir!



**#PracegoVer:** na imagem, existem as mãos de um programador com um relógio de pulso digitando no teclado de um notebook aberto, o qual está em cima de uma mesa.

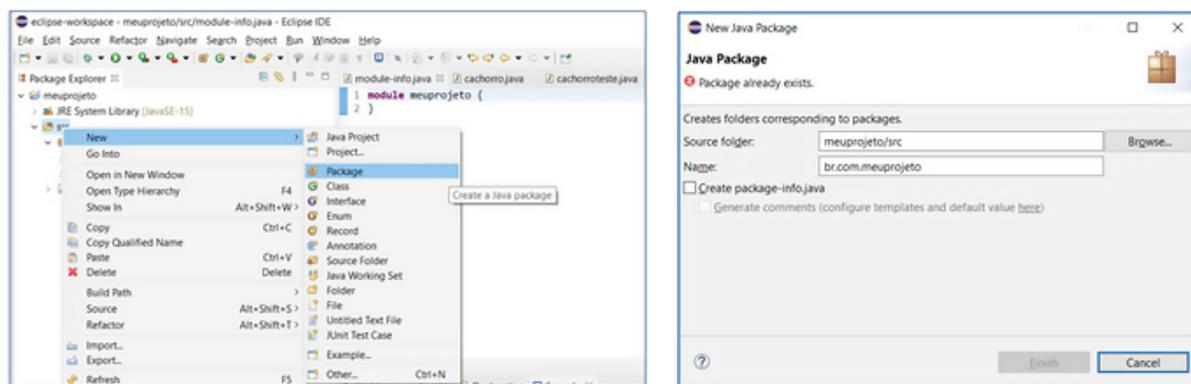
### Elaborar pacote

No caso de desejarmos criar classes que realizam cálculos estatísticos, podemos elaborar um pacote e chamá-lo de "estatísticas".

### Utilizar pacote

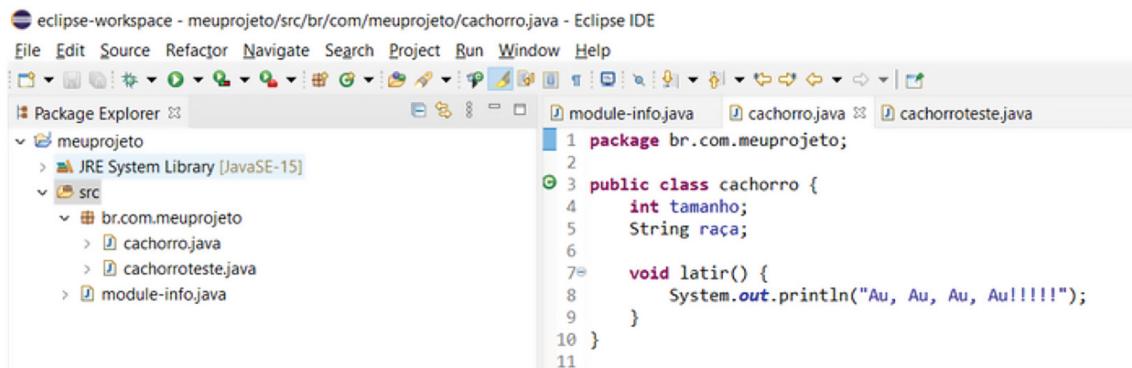
O pacote em questão armazenará todas as classes. Desse modo, quem necessitar utilizar o pacote, basta acioná-lo em seu programa.

É bem simples criar um pacote, observe na imagem abaixo:



**#PraCegoVer:** na imagem, temos um *print* da tela do software Eclipse. Do lado esquerdo, existe o *dashboard* do programa com o menu de contexto acionado, destacando a opção “package”. Do lado direito, existe a aba de criação aberta, indicando as opções de configuração do pacote que será criado.

Após realizar os procedimentos acima, note, na próxima imagem, que o nome do pacote foi criado/adicionado ao código que já você tinha pronto.



**#PraCegoVer:** na imagem, temos um *print* da tela do software Eclipse com *dashboard*, barra de ferramenta na parte superior, com o layout de pastas no canto esquerdo e com o painel de programação no lado direito, em que está escrito o seguinte código:

```
package br.com.meuprojeto; public class cachorro { int tamanho; String
raça; void latir() { System.out.println("Au, Au, Au, au!!!!"); } }.
```

Neste tópico, você estudou sobre o conceito de pacote e como criá-lo no IDE. Siga adiante para conhecer outro elemento importante para o seu futuro profissional enquanto programador que usa a linguagem Java: **o método com parâmetros**.

## Método com parâmetros

Imagine uma situação em que você foi contratado para construir uma aplicação bancária. A primeira classe a ser desenvolvida é a Conta, com os atributos **cliente** e **saldo**, onde deve ser possível realizar operações como exibir saldo, sacar, depositar e transferir um valor. Veja a seguir!

Conta
cliente: String
saldo: double
exibeSaldo() : void
saca(valor double): void
deposita(valor: double): void
transfere Para(destino : Conta, valor: double): void.

**#PraCegoVer:** na imagem, temos um quadro em que há a descrição da classe com seus atributos. Nela, existe o seguinte código:

```
Conta cliente : String saldo : double exibeSaldo() : void saca(valor double): void deposita(valor: double): void transfere Para(destino : Conta, valor double): void.
```

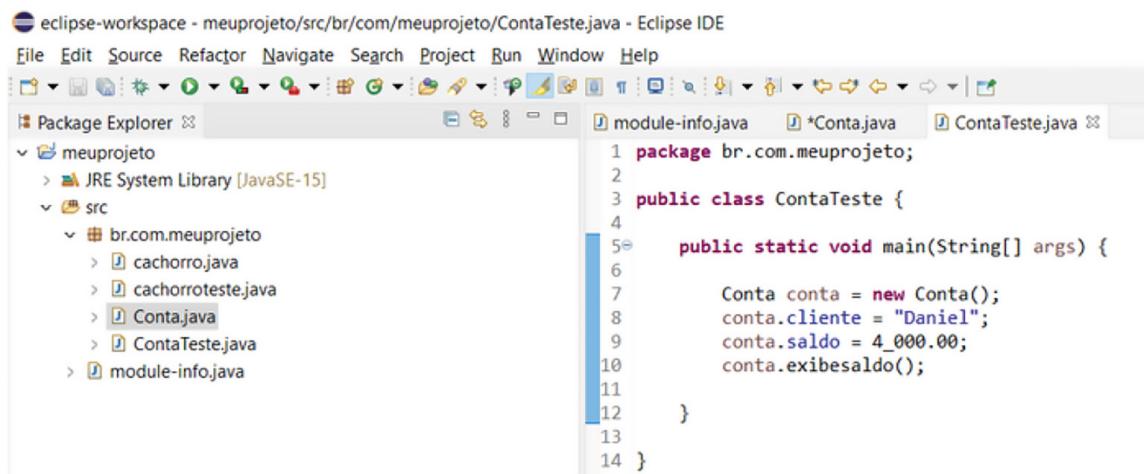
O primeiro passo, então, é criar a classe em questão.

```
eclipse-workspace - meuprojeto/src/br/com/meuprojeto/Conta.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
File View Insert Properties Run Project Clean Preferences Help
Package Explorer module-info.java *Contajava ContaTeste.java
meuprojeto JRE System Library [JavaSE-15]
src
  br.com.meuprojeto
    cachorro.java
    cachorroteste.java
    Contajava
    ContaTeste.java
    module-info.java
1 package br.com.meuprojeto;
2
3 public class Conta {
4
5   String cliente;
6   double saldo;
7
8   void exibeSaldo() {
9     System.out.println(cliente + " seu saldo é " + saldo);
10  }
11
12 }
```

**#PraCegoVer:** na imagem, temos um *print* da tela do software Eclipse com *dashboard*, barra de ferramenta na parte superior, com o layout de pastas no canto esquerdo e com o painel de programação no lado direito, em que está escrito o seguinte código:

```
package br.com.meuprojeto; public class Conta { String cliente; double saldo; void exibeSaldo() { System.out.println(cliente " seu saldo é " saldo);}}.
```

Em seguida, você precisa criar outra classe, agora chamada de ContaTeste, selecionando a opção “*public Static void main (Strings[] args)*”. Observe na imagem, como fica esse passo realizado:



The screenshot shows the Eclipse IDE interface. The top menu bar includes File, Edit, Source, Refactor, Navigate, Search, Project, Run, Window, and Help. Below the menu is a toolbar with various icons. On the left, the Package Explorer shows a project named 'meuprojeto' with a 'src' folder containing files: 'JRE System Library [JavaSE-15]', 'cachorro.java', 'cachorroteste.java', 'Contajava', 'ContaTeste.java', and 'module-info.java'. The right side of the interface displays the code for 'ContaTeste.java' in a text editor:

```

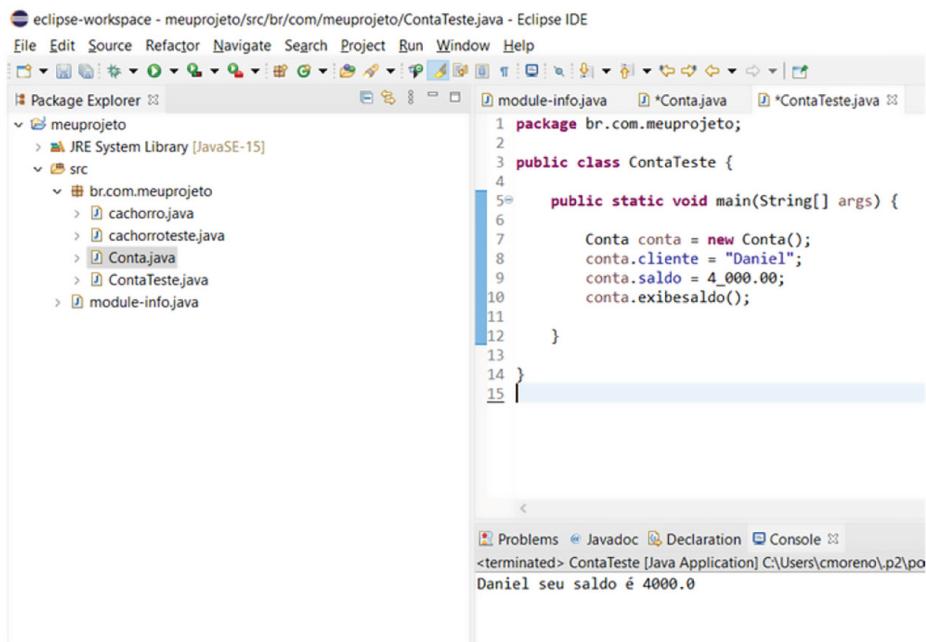
1 package br.com.meuprojeto;
2
3 public class ContaTeste {
4
5     public static void main(String[] args) {
6
7         Conta conta = new Conta();
8         conta.cliente = "Daniel";
9         conta.saldo = 4_000.00;
10        conta.exibesaldo();
11    }
12
13
14 }

```

**#PraCegoVer:** na imagem, temos um *print* da tela do software Eclipse com *dashboard*, barra de ferramenta na parte superior, com o layout de pastas no canto esquerdo e com o painel de programação no lado direito, em que está escrito o seguinte código:

```
package br.com.meuprojeto; public class ContaTeste { public static void main(String[] args) { Conta conta = new Conta(); conta.cliente = "Daniel"; conta.saldo = 4_000.00; conta.exibesaldo();}}
```

Ao executar seu programa, confira se o resultado fica conforme apresentado abaixo.



The screenshot shows the Eclipse IDE interface with the 'Console' tab selected at the bottom. The output window displays the following text:

```
<terminated> ContaTeste [Java Application] C:\Users\cmoreno\p2\po
Daniel seu saldo é 4000.0
```

**#PraCegoVer:** na imagem, temos um *print* da tela do software Eclipse com *dashboard*, barra de ferramenta na parte superior, com o layout de pastas no canto esquerdo e com o painel de programação no lado direito, em que está escrito o seguinte código:

```
package br.com.meuprojeto; public class ContaTeste { public static void main(String[] args) { Conta conta = new Conta(); conta.cliente = "Daniel"; conta.saldo = 4_000.00; conta.exibesaldo();}} Além disso, existe a classe ContaTeste sendo executada. Na aba inferior denominada "Console", está escrito "Daniel seu saldo é 4000.0"
```

Para implementar o método que realiza o saque de dinheiro, será necessário observar algumas premissas, confira na sequência!

### 1 - Autorização

Verificar se há autorização para o saque.

### 2 - Limite diário

Verificar se o limite do dia não foi ultrapassado.

### 3 - Saldo

Verificar se a conta tem saldo.

### 4 - Atualizar conta

Deduzir o valor da conta após o saque.

### 5 - Atualizar registros

Atualizar os registros com o novo valor da conta.

### 6 - Entrega

Entregar o dinheiro para o cliente.

Agora, assista ao vídeo com o passo a passo para entender como adicionar os métodos de saque, depósito e “*this*”.



## Vídeo

Confira o [vídeo](#) de como adicionar os métodos saque, depósito e “this”.

Perdeu algum detalhe? Confira o que foi abordado no vídeo.

Olá! Na construção de uma aplicação bancária, após criar as classes e seus atributos é necessário implementar o método. Aqui, vamos implantar um método para realizar o saque de dinheiro.

Comece pela classe “Conta” adicionando o método de saque.

```
package br.com.meuprojeto;

public class Conta {

    String cliente;
    double saldo;

    void exibesaldo() {
        System.out.println(cliente + " seu saldo é " +
                           saldo);
    }

    void saque(double valor) {
        saldo -= valor;
    }
}
```

Já na classe “ContaTeste”, é necessário realizar a chamada do método de saque. Isso pode ser feito simulando um saque no valor de R\$ 2.000,00, por exemplo.

```
package br.com.meuprojeto;

public class ContaTeste {

    public static void main(String[] args) {

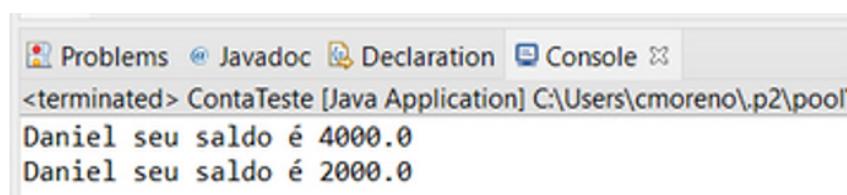
        Conta conta = new Conta();
        conta.cliente = "Daniel";
        conta.saldo = 4_000.00;
        conta.exibesaldo();

        conta.saque(2000);
        conta.exibesaldo();

    }
}
```

Observe que a linha de código é ajustada para esse fim.

Na sequência conseguimos observar o método de saque realizado.



```
Problems @ Javadoc Declaration Console
<terminated> ContaTeste [Java Application] C:\Users\cmoreno\p2\pool
Daniel seu saldo é 4000.0
Daniel seu saldo é 2000.0
```

Neste momento, é implantado o método de depósito. Para isso, é adicionado o método na classe “Conta”.

```
public class Conta {

    String cliente;
    double saldo;

    void exibesaldo() {
        System.out.println(cliente + " seu saldo é " +
                           saldo);
```

```
}

void saque(double valor) {
    saldo -= valor;

}

void deposita(double valor) {
    saldo += valor;

}
```

Depois, adiciona-se a chamada do método na classe “ContaTeste”.

```
public class ContaTeste {

    public static void main(String[] args) {

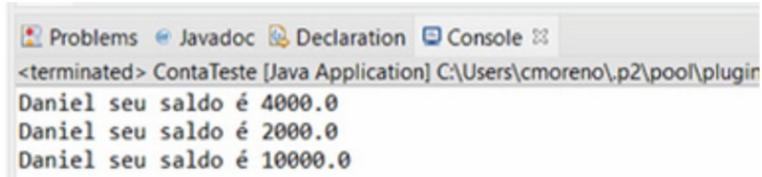
        Conta conta = new Conta();
        conta.cliente = "Daniel";
        conta.saldo = 4_000.00;
        conta.exibesaldo();

        conta.saque(2000);
        conta.exibesaldo();

        conta.deposita(8000);
        conta.exibesaldo();

    }
}
```

Na sequência, executamos e são exibidos os resultados.



```
Problems Javadoc Declaration Console
<terminated> ContaTeste [Java Application] C:\Users\cmoreno\p2\pool\plugins
Daniel seu saldo é 4000.0
Daniel seu saldo é 2000.0
Daniel seu saldo é 10000.0
```

Por último, é criado um método que realizará a transferência de valor para uma conta de destino, o qual será implantado na classe “Conta”.

Observe que o método *“this.”*, serve para fazer a referência do objeto.

```
package br.com.meuprojeto;

public class Conta {

    String cliente;
    double saldo;

    void exibesaldo() {
        System.out.println(cliente + " seu saldo é " +
                           saldo);

    }

    void saque(double valor) {
        saldo -= valor;

    }

    void deposita(double valor) {
        saldo += valor;

    }

    void transferenciaValor(Conta destino, double
                           valor) {
        this.saque(valor);
        destino.deposita(valor);

    }
}
```

Seguindo com o processo, implante a classe “ContaTeste”, comentando os métodos “conta.saque” e “conta.deposita”.

```
package br.com.meuprojeto;

public class ContaTeste {

    public static void main(String[] args) {

        Conta conta = new Conta();
        conta.cliente = "Daniel";
        conta.saldo = 4 _ 000.00;
        conta.exibesaldo();

        //conta.saque(2000);
        //conta.exibesaldo();

        //conta.deposita(8000);
        //conta.exibesaldo();

        Conta destino = new Conta();
        destino.cliente = "Carlos";
        destino.saldo = 5 _ 000.00;
        destino.exibesaldo();

    }
}
```

Dessa maneira, chegamos ao resultado da transferência.

Agora é com você. Pratique esse aprendizado!

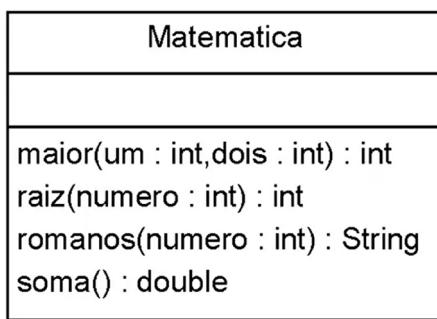
Neste tópico, a partir do exemplo de software que simula uma aplicação bancária, você observou como o método com parâmetros é aplicado. Contudo, existem situações específicas que vão além da apresentada neste tópico.

Sobre o que acabamos de abordar, lançamos o seguinte questionamento: e quanto ao método com retorno? Você saberia explicar do que se trata? Você verá sobre ele logo na sequência.

## Método com retorno

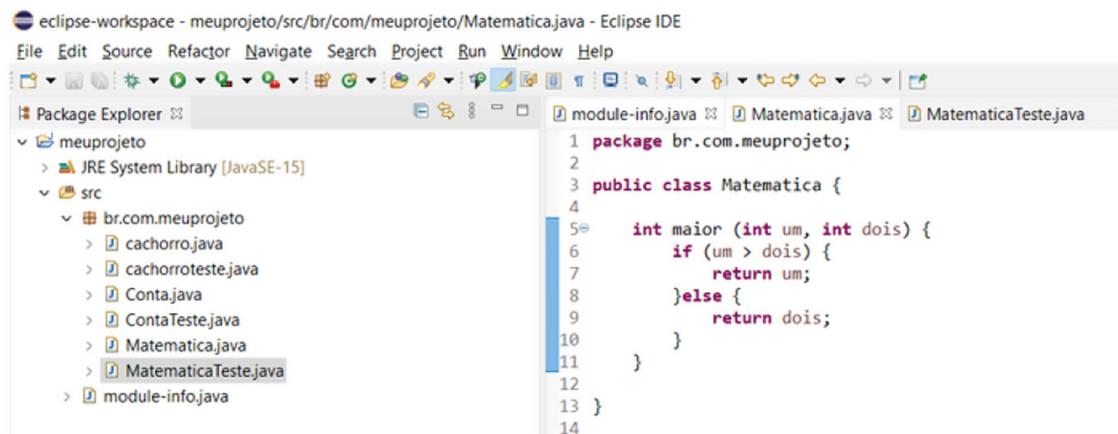
Até aqui você viu como executar o método que não retornava valor, por isso, é conhecido como “**void**”, mas, em algumas situações profissionais futuras em que você desenvolverá softwares mais complexos, o método precisa retornar algum valor.

Para que você entenda de forma mais objetiva, vamos utilizar o exemplo em que aparece a classe Matemática, com os métodos de maior, raiz, romanos e soma, que precisam retornar valores. Assim como no caso de necessitarmos desses valores para continuar as operações.



**#PraCegoVer:** na imagem, temos uma classe com seus atributos. Trata-se da classe Matemática com suas operações de maior, raiz, romanos e soma. Além disso, existe um código escrito da seguinte maneira: Matematica: maior(um : int, dois : int) : int raiz(numero: int) : int romanos(numero: int) : String soma(): double.

Para criar um método com retorno, precisamos criar duas classes, uma **classe intitulada Matemática** e outra chamada MatematicaTeste. Para isso, selecione a opção “*public Static void main (Strings[] args)*”, conforme você pode acompanhar nas duas imagens a seguir.



eclipse-workspace - meuprojeto/src/br/com/meuprojeto/Matematica.java - Eclipse IDE

```

File Edit Source Refactor Navigate Search Project Run Window Help
Package Explorer  module-info.java  Matematica.java  MatematicaTeste.java
meuprojeto
  JRE System Library [JavaSE-15]
src
  br.com.meuprojeto
    cachorro.java
    cachorroteste.java
    Conta.java
    ContaTeste.java
    Matematica.java
    MatematicaTeste.java
  module-info.java

```

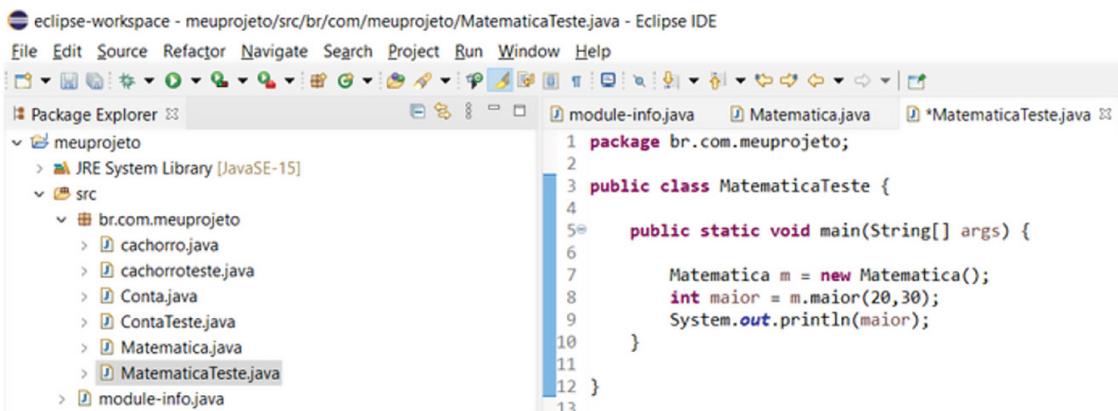
```

1 package br.com.meuprojeto;
2
3 public class Matematica {
4
5     int maior (int um, int dois) {
6         if (um > dois) {
7             return um;
8         }else {
9             return dois;
10        }
11    }
12 }
13
14

```

**#PraCegoVer:** na imagem, temos um *print* da tela do software Eclipse com *dashboard*, barra de ferramenta na parte superior, com o layout de pastas no canto esquerdo e com o painel de programação no lado direito, em que está escrito o seguinte código:

```
package br.com.meuprojeto; public class Matematica int maior (int um, int dois) { if (um > dois) { return um; }else { return dois;}}
```



eclipse-workspace - meuprojeto/src/br/com/meuprojeto/MatematicaTeste.java - Eclipse IDE

```

File Edit Source Refactor Navigate Search Project Run Window Help
Package Explorer  module-info.java  Matematica.java  *MatematicaTeste.java
meuprojeto
  JRE System Library [JavaSE-15]
src
  br.com.meuprojeto
    cachorro.java
    cachorroteste.java
    Conta.java
    ContaTeste.java
    Matematica.java
    MatematicaTeste.java
  module-info.java

```

```

1 package br.com.meuprojeto;
2
3 public class MatematicaTeste {
4
5     public static void main(String[] args) {
6
7         Matematica m = new Matematica();
8         int maior = m.maior(20,30);
9         System.out.println(maior);
10    }
11 }
12
13

```

**#PraCegoVer:** na imagem, temos um *print* da tela do software Eclipse com *dashboard*, barra de ferramenta na parte superior, com o layout de pastas no canto esquerdo e com o painel de programação no lado direito, em que está escrito o seguinte código:

```
package br.com.meuprojeto; public class MatematicaTeste public static void main(String[] args) { Matematicam = new Matematica(); int maior = m.maior(20,30); System.out.println(maior);}}
```

No exemplo abaixo, implantamos o método **maior**, que verificará qual será o maior entre dois números. Acompanhe!

```

eclipse-workspace - meuprojeto/src/br/com/meuprojeto/MatematicaTeste.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
Package Explorer JRE System Library [JavaSE-15]
meuprojeto
src
  br.com.meuprojeto
    Cachorro.java
    CachorroTeste.java
    Conta.java
    ContaTeste.java
    Matematica.java
    MatematicaTeste.java
  module-info.java
  1 package br.com.meuprojeto;
  2
  3 public class MatematicaTeste {
  4
  5     public static void main(String[] args) {
  6
  7         Matematica m = new Matematica();
  8         int maior = m.maior(20,30);
  9         System.out.println(maior);
 10     }
 11
 12 }
 13
 14
  
```

Problems Javadoc Declaration Console <terminated> MatematicaTeste [Java Application] C:\Users\cmoreno\p2\pool\pli  
30

**#PraCegoVer:** na imagem, temos um *print* da tela do software Eclipse com *dashboard*, barra de ferramenta na parte superior, com o layout de pastas no canto esquerdo e com o painel de programação no lado direito, em que está escrito o seguinte código:

```
package br.com.meuprojeto; public class MatematicaTeste public static void main(String[] args) { Matematicam = new Matematica(); int maior = m.maior(20,30); System.out.println(maior);} 
```

Na aba inferior denominada “Console”, está escrito “30”.

Em seguida, implantaremos o método que executa a **soma** entre dois números.

```

eclipse-workspace - meuprojeto/src/br/com/meuprojeto/Matematica.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
Package Explorer JRE System Library [JavaSE-15]
meuprojeto
src
  br.com.meuprojeto
    Cachorro.java
    CachorroTeste.java
    Conta.java
    ContaTeste.java
    Matematica.java
    MatematicaTeste.java
  module-info.java
  1 package br.com.meuprojeto;
  2
  3 public class Matematica {
  4
  5     int maior (int um, int dois) {
  6         if (um > dois) {
  7             return um;
  8         } else {
  9             return dois;
 10         }
 11     }
 12
 13     double soma(double um, double dois) {
 14         double s = um + dois;
 15         return s;
 16     }
 17
 18 }
  
```

**#PraCegoVer:** na imagem, temos um *print* da tela do software Eclipse com *dashboard*, barra de ferramenta na parte superior, com o layout de pastas no canto esquerdo e com o painel de programação no lado direito, em que está escrito o seguinte código:

```
package br.com.meuprojeto; public class Matematica { int maior (int um, int dois) { if (um > dois) { return um; } else { return dois; } } double soma(double um, double dois) { double s = um + dois; return s; } } 
```

Ao executar seu programa, o resultado será a soma entre os dois números fornecidos, confira na imagem a seguir:

```

eclipse-workspace - meuprojeto/src(br.com.meuprojeto)/MatematicaTeste.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
Package Explorer JRE System Library [JavaSE-15]
meuprojeto
src
  br.com.meuprojeto
    cachorro.java
    cachorroteste.java
    Conta.java
    ContaTeste.java
    Matematica.java
    MatematicaTeste.java
  module-info.java
  module-info.java
  Matematica.java
  MatematicaTeste.java

1 package br.com.meuprojeto;
2
3 public class MatematicaTeste {
4
5     public static void main(String[] args) {
6
7         Matematica m = new Matematica();
8         int maior = m.maior(20,30);
9         System.out.println(maior);
10
11         double som = m.soma(50,60);
12         System.out.println(som);
13     }
14 }
15
16
17

```

Console

```

30
110.0

```

**#PraCegoVer:** na imagem, temos um *print* da tela do software Eclipse com *dashboard*, barra de ferramenta na parte superior, com o layout de pastas no canto esquerdo e com o painel de programação no lado direito, em que está escrito o seguinte código:

```

package br.com.meuprojeto; public class MatematicaTeste public static
void main(String[] args) { Matematica m = new Matematica(); int maior =
m.maior(20,30); System.out.println(maior); double som = m.soma(50,60);
System.out.println(som);}

```

Na aba inferior denominada “Console”, está escrito “30 e 110.0”.

Deste modo, ainda tendo como base o nosso exemplo de software de aplicação bancária, essa é a maneira pela qual o método com retorno é feito. Todavia, devemos abordar outros dois métodos relevantes para o seu futuro profissional, os métodos *get()* e *set()* (encapsulamento). Vamos conhecê-los!

## Métodos *get ()* e *set ()* (encapsulamento)

É importante proteger as variáveis, e para isso utilizamos o encapsulamento, desse modo ninguém poderá configurá-las com um valor inapropriado. Inclusive, encapsulamos variáveis para evitarmos o excesso de códigos.

```

        item = el->FirstChildElement(); item = item->getNextElement();
        string el_name = item->Attribute( "name" );
        string type = item->Attribute( "type" );
        type == "quarto" )

        std::string item_name = item->Attribute( "name" );
        std::string spritename = item->Attribute( "spritename" );
        float x = boost::lexical_cast<float>( item->Attribute( "x" ) );
        float y = boost::lexical_cast<float>( item->Attribute( "y" ) );
        float offset = boost::lexical_cast<float>( item->Attribute( "offset" ) );

        spriteDescList::iterator sp = sprite_descs.begin();
        for( ; sp != sprite_descs.end(); ++sp )
            if ( sp->name_ == spritename )
                break;
        if ( sp == sprite_descs.end() )
            throw "error";
    }
}

```

A maioria dos valores utilizados nas classes são codificadas com definições de limites. Podemos citar, como exemplo, algo que não funcionaria com valores negativos, como a quantidade de quartos em uma casa, afinal, não há quarto com a quantidade negativa, certo?

### #PraCegoVer

Na imagem, há uma tela de computador com uma série de linhas de código de programação.

A velocidade de um carro também entra na lista de exemplos de classes codificadas com definições de limites, pois ela não apresenta valores negativos, somente indica 50 km/h, 80 km/h, e assim por diante.



### #PraCegoVer

Na imagem, um painel de um carro está em destaque, marcando duzentos quilômetros por hora.

Para fixar melhor esses conceitos, veja na prática, como são utilizados os métodos `get()` e `set()`, os quais servem justamente para realizar esse encapsulamento.

```
public class Funcionario {  
    private String nome;  
    private boolean ativo;  
  
    public String getNome() {  
        return nome;  
    }  
  
    public void setNome (String n) {  
        this.nome = n;  
    }  
    public boolean isAtivo; {  
        return ativo;  
    }  
    public void setAtivo(boolean ativo){  
        this.ativo = ativo;  
    }  
}
```

Assim que outros códigos forem forçados a passar por métodos de configuração, temos como validar se o parâmetro chamado será viável. Dessa maneira, garantimos o uso somente de valores válidos!

Porém, ainda temos uma outra situação no contexto de desenvolvimento de softwares que se relaciona com a transmissão de atributos e métodos em classes diferentes, estamos falando do conceito de herança, veja a seguir.

## Herança

Para falarmos sobre o assunto de Herança precisamos destacar a importância deste relacionamento, trazendo como exemplo a situação do exemplo abaixo.

A herança consiste no relacionamento “É-Um”. Imagine, por exemplo, que você construirá um programa de simulação de animais que contemplará uma longa série de espécies. Porém, para facilitar seu entendimento, vamos utilizar apenas três tipos de animais: um cachorro, uma coruja e uma galinha.

Como sabemos, esses animais trazem características em comum, mas apresentarão atributos e métodos diferentes, observe a seguir:

### Características em comum

Todos os animais dormem, comem, movimentam-se e fazem barulho.

### Particularidades

Um cachorro enterra o osso, a coruja voa, enquanto a galinha bota ovos.

A herança, então, nos permite reaproveitar os códigos.

Pense na dificuldade que você teria em manter uma aplicação com 100 animais ou mais, em que todos teriam atributos semelhantes e precisasse alterar o método de dormir?

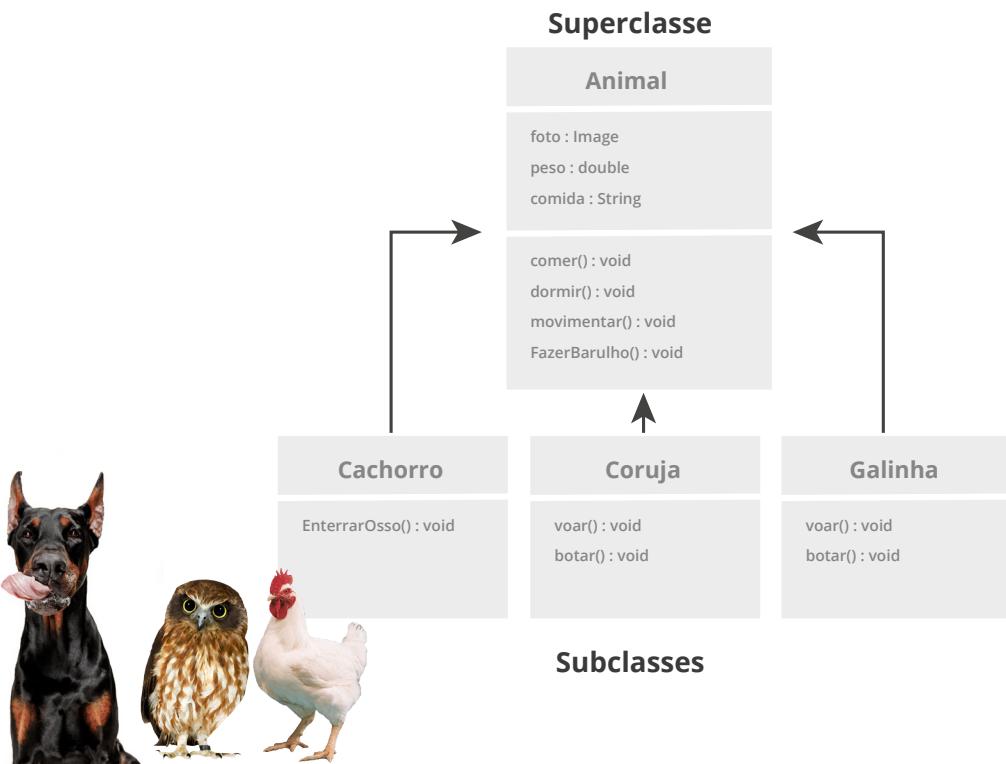
Cachorro	Coruja	Galinha
foto : Image peso : double comida : String	foto : Image peso : double comida : String	foto : Image peso : double comida : String
comer() : void dormir() : void movimentar() : void FazerBarulho() : void EnterrarOsso() : void	comer() : void dormir() : void movimentar() : void FazerBarulho() : void voar() : void botar() : void	comer() : void dormir() : void movimentar() : void FazerBarulho() : void voar() : void botar() : void





**#PraCegoVer:** na imagem, temos um esquema exemplificando o conceito de herança nas classes Cachorro, Coruja e Galinha, as quais trazem seus próprios atributos. Abaixo existem três colunas nas quais cada uma tem os seguintes códigos: Cachorro: foto: Image; peso: double; comida: String; comer(): void; dormir(): void; movimentar(): void; FazerBarulho(): void; EnterrarOsso(): void; Coruja: foto: Image; peso: double; comida: String; comer(): void; dormir(): void; movimentar(): void; FazerBarulho(): void; voar(): void; botar(): void; Galinha: foto: Image; peso: double; comida: String; comer(): void; dormir(): void; movimentar(): void; FazerBarulho(): void; voar(): void; botar(): void.

Por essa lógica, a herança nos propicia a possibilidade de criar **superclasses** para que recebam esses atributos e métodos semelhantes. Neste caso, seria criada uma classe Animal, em que o cachorro, a coruja e a galinha poderiam ser filhas dessa classe, ou seja, **subclasses**. Com isso, dizemos que um cachorro “É-Um” animal, a coruja “É-Um” animal e a galinha “É-Um” animal.



**#PraCegoVer:** na imagem, temos um esquema exemplificando uma superclasse (Animal) ligada a suas subclasses (Cachorro, Coruja e Galinha). além disso, existem três colunas nas quais cada uma tem os seguintes códigos: Superclasse Animal foto: Image peso: double comida : String comer(): void dormir(): void movimentar(): void fazerBarulho() : void. Abaixo há três colunas, que são as subclasses: Cachorro EnterrarOsso() : void; Coruja voar () : void botar() : void; Galinha voar() : void botar() : void

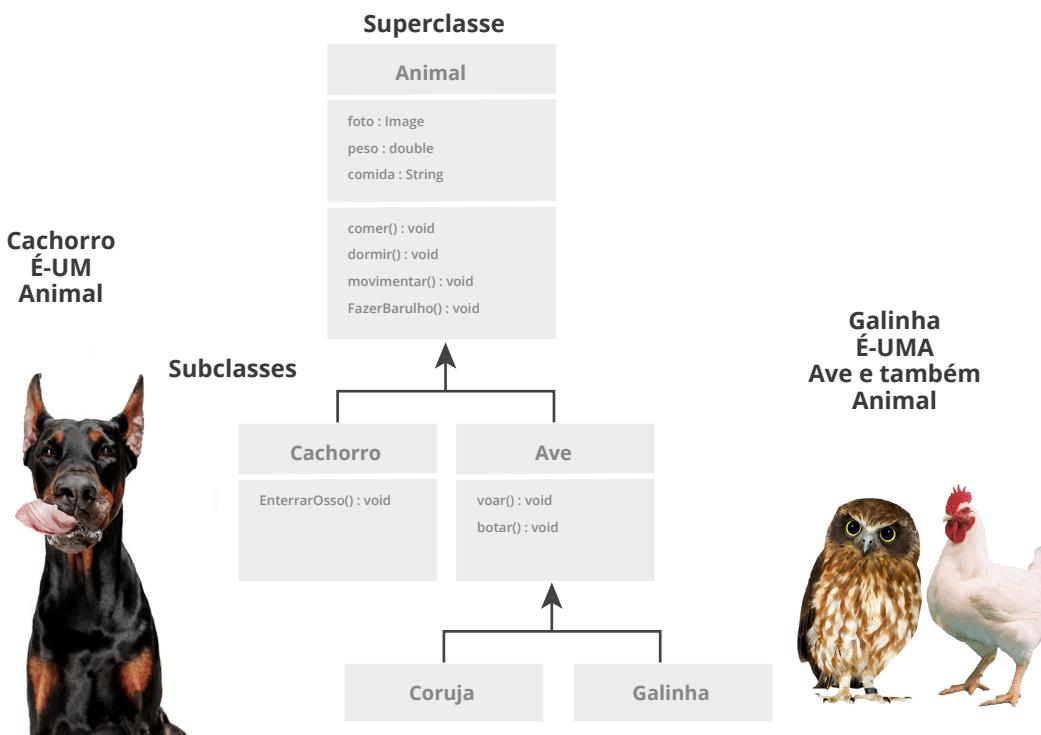
Além disso, é essencial considerar dois aspectos importantes.  
Descubra quais são eles!

### Criação de outro nível de superclasse

Tanto a coruja quanto a galinha compartilham as mesmas características, por isso, será criado outro nível de superclasse, o qual será chamado Ave. Este receberá as características semelhantes.

***extends***

Observe que a classe Cachorro estende a superclasse Animal, ou seja, utilizamos *extends* para transformar uma classe em uma subclasse de outra classe.



**#PraCegoVer:** na imagem, temos um esquema exemplificando uma superclasse (Animal) ligada a suas subclasses (Cachorro e Ave). Da subclasse Ave, ainda temos Coruja e Galinha. No caso, o cachorro É-UM animal, enquanto a galinha É-UMA ave e um animal. Além disso, existe um fluxograma com os seguintes códigos: Superclasse Animal foto: Image peso : double comida : String comer() : void dormir : void movimentar : void fazerBarulho() : void Galinha É-UMA Ave e também Animal Cachorro Ave enterraroso() : void voar : void botar(): void Subclasses Cachorro É-UM Animal Coruja Galinha

Quando uma classe herda de outra, dizemos que ela pode utilizar ou passar a ter todos os seus métodos e atributos.

Vamos ver esses conceitos na prática? Assista ao vídeo para ver como é simples criar uma classe que representa toda essa estrutura.



## Vídeo

Confira o [vídeo](#) de como aplicar herança.

Perdeu algum detalhe? Confira o que foi abordado no vídeo.

Olá! Chegou o momento de praticar como aplicar a herança.

O primeiro passo é elaborar um pacote chamado “br.com.heranca”.

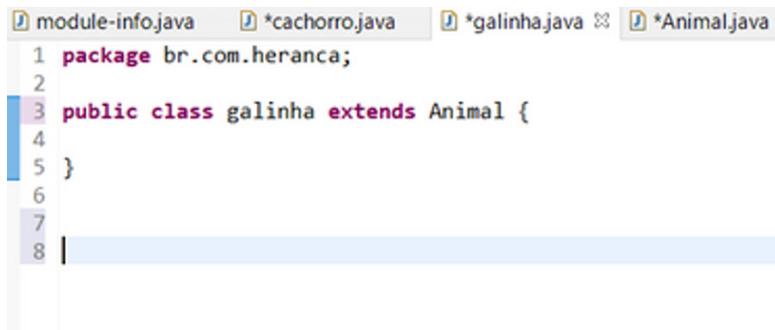
Em seguida, crie outras duas classes dentro desse pacote: Cachorro e Galinha.

```
module-info.java  cachorro.java  galinha.java
1 package br.com.heranca;
2
3 public class galinha {
4
5 }
6
```

Na sequência, crie mais uma classe, a qual contemplará as mesmas características das classes Cachorro e Galinha, agora com o nome Animal.

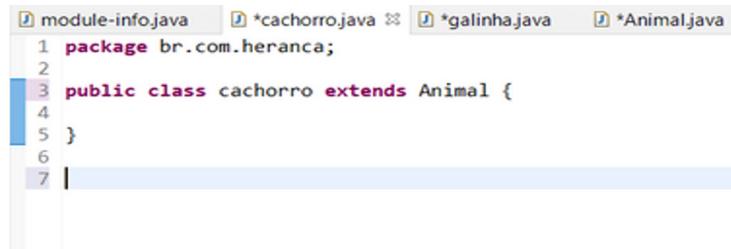
```
module-info.java  *cachorro.java  *galinha.java  *Animal.java
1 package br.com.heranca;
2
3 public class Animal {
4
5     double peso;
6     String comida;
7
8     void dormi() {System.out.println("Dormiu");}
9     void fazerBarulho() {System.out.println("Fazer barulho");}
10
11 }
12
```

Agora, estenda a classe Galinha para a classe Animal.



```
module-info.java *cachorro.java *galinha.java *Animal.java
1 package br.com.heranca;
2
3 public class galinha extends Animal {
4
5 }
6
7
8 |
```

Depois, é necessário estender a classe Cachorro para a classe Animal também.



```
module-info.java *cachorro.java *galinha.java *Animal.java
1 package br.com.heranca;
2
3 public class cachorro extends Animal {
4
5 }
6
7 |
```

O próximo passo é criar uma classe chamada AnimalTeste, selecionando a opção “*public static void main (String[] args)*”.



```
module-info.java *cachorro.java *galinha.java *Animal.java AnimalTeste.java
1 package br.com.heranca;
2
3 public class AnimalTeste {
4
5     public static void main(String[] args) {
6         // TODO Auto-generated method stub
7     }
8 }
9
10
11 |
```

Aqui é possível implementar a classe, conforme indicado.

Pronto! Agora você tem uma herança aplicada.

No vídeo, a subclasse Cachorro e a subclasse Galinha herdaram as características e os métodos da superclasse Animal. Ambas, então, utilizam desses atributos como se estivessem inseridos diretamente na superclasse.

Agora, como identificar se uma classe pode herdar de outra?

Na verdade, é bem simples! Primeiramente, você precisa saber que a herança funciona sempre em uma única direção, sendo que o teste do “É-Um” o auxilia nessa identificação. Veja alguns exemplos apresentados a seguir!

### Exemplo 1

O quadrado é uma forma? Sim!

### Exemplo 2

A cerveja é uma bebida? Sim!

### Exemplo 3

O forno é uma cozinha? Não!

### Exemplo 4

O funcionário é uma pessoa? Sim!

Com isso, é possível identificar quando uma classe deveria estender ou não outra classe, aproveitando os atributos comuns a todos.

Parabéns! Você chegou ao final do módulo 2.

A partir dos exemplos oriundos do IDE, você conheceu sobre a aplicação dos pacotes Java Lang e Java Util, bem como a aplicação do método com parâmetros, retorno e `get()` e `set()` e de herança. Inclusive, todos esses são muito relevantes, sobretudo, em seu futuro ambiente profissional.

No próximo módulo, você estudará outros pontos que merecem sua atenção, além de serem muito recorrentes em contextos de desenvolvimento de programas usando a linguagem Java. Estamos nos referindo ao tratamento de exceções, interface gráfica, eventos e integração. Vamos lá?



Módulo 3

# Tratamento de exceções, interface gráfica, eventos e integração

---

# Tratamento de exceções, interface gráfica, eventos e integração

Neste módulo, você verá o tratamento de exceções e como implementá-los no código. Descobrirá, ainda, como criar uma interface gráfica simples e entenderá os conceitos relacionados aos *layouts*. Ao final, você compreenderá o conceito de eventos em Java, sendo que tais aspectos são imprescindíveis para a sua atuação enquanto programador. Vamos começar?

## Tratamento de exceções

Em primeiro lugar é importante saber sobre o tratamento de exceções, que consiste em um recurso utilizado para construir programas robustos e tolerantes a falhas na linguagem Java. Entenda melhor!

### Exceção

Uma **exceção** consiste na indicação de um problema que surge durante a execução de um programa.

### Nome “exceção”

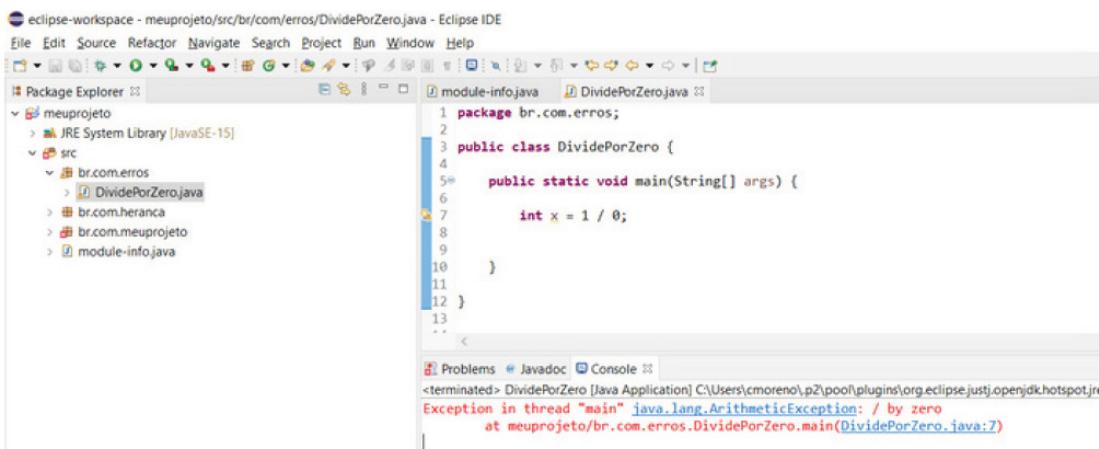
O nome “**exceção**” significa que o problema não é frequente, ou seja, trata-se de uma exceção à regra.

Com o tratamento de exceções, um programa pode continuar executando em vez de simplesmente encerrar, contribuindo para a sua qualidade. No próximo tópico, você como ele funciona na prática! Confira!

## Exception

Quando se usa a linguagem Java, sobretudo, em desenvolvimentos de softwares, podem ocorrer erros de lógica nas linhas de programação, ou ainda alguma inconsistência no momento de acessar dispositivos externos, fatos raros de se acontecer, ou seja, exceções (daí o nome do termo *exception*).

Diante disso, vamos aprender como lidar com *exception*, evitando que isso ocorra. Adiante, será criado um pacote chamado “br.com.erros”. Em seguida, será elaborada uma classe denominada DividePorZero.



The screenshot shows the Eclipse IDE interface. The top menu bar includes File, Edit, Source, Refactor, Navigate, Search, Project, Run, Window, and Help. The left sidebar displays the Package Explorer with a project named 'meuprojeto' containing packages 'br.com.erros' (with 'DividePorZero.java'), 'br.com.heranca', and 'br.com.meuprojeto', along with 'src' and 'module-info.java'. The right panel shows the code editor with 'DividePorZero.java' open, containing the following code:

```
1 package br.com.erros;
2
3 public class DividePorZero {
4
5     public static void main(String[] args) {
6
7         int x = 1 / 0;
8
9     }
10}
11
12}
```

Below the code editor is the Console view, which shows the output of a terminated Java application:

```
<terminated> DividePorZero [Java Application] C:\Users\cmoreno\p2\pool\plugins\org.eclipse.jdt.openjdk.hotspot.jre.win32.x86_64_15.0.0.v20210615_1000\jre\bin\java.exe DividePorZero
Exception in thread "main" java.lang.ArithmaticException: / by zero
        at meuprojeto.br.com.erros.DividePorZero.main(DividePorZero.java:7)
```

**#PraCegoVer:** na imagem, temos um *print* da tela do software Eclipse com *dashboard*, barra de ferramenta na parte superior, com o layout de pastas no canto esquerdo e com o painel de programação no lado direito, em que está escrito o seguinte código:

```
package br.com.erros; public class DividePorZero { public static void main(String[] args) { int x = 1/;}}
```

Agora, vamos executar o código com o auxílio do programa. **Veja que, no console, há a possibilidade de inserir os números que desejamos para utilizar no cálculo.** Ao final, execute o programa e confira se o resultado é o mesmo apresentado na imagem abaixo.

```

eclipse-workspace - meuprojeto/src(br.com.erros.DividePorZero.java) - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
Package Explorer module-info.java DividePorZero.java
meuprojeto JRE System Library [JavaSE-15]
src br.com.erros
    DividePorZero.java
    br.com.heranca
    br.com.meuprojeto
    module-info.java
1 package br.com.erros;
2
3 import java.util.Scanner;
4
5 public class DividePorZero {
6
7     public static void main(String[] args) {
8
9         Scanner s = new Scanner(System.in);
10        System.out.println("Número: ");
11
12        int a = s.nextInt();
13        System.out.println("Divisor: ");
14
15        int b = s.nextInt();
16        System.out.println(a / b);
17    }
18
19 }
20
21
22
Problems Javadoc Console
<terminated> DividePorZero [Java Application] C:\Users\cmoreno\p2\pool\plugins\org.eclipse.jdt.openjdk.hotspot.jre.full.win32.x86_64
Número:
1
Divisor:
0
Exception in thread "main" java.lang.ArithmeticException: / by zero
at meuprojeto.br.com.erros.DividePorZero.main(DividePorZero.java:16)

```

**#PraCegoVer:** na imagem, temos um *print* da tela do software Eclipse com *dashboard*, barra de ferramenta na parte superior, com o layout de pastas no canto esquerdo e com o painel de programação no lado direito, em que está escrito o seguinte código:

```

package br.com.erros; import java.util.Scanner; public class DividePorZero
public static void main(String[] args) { Scanners new Scanner(System.
in); System.out.println("Número: "); int a.s.nextInt(); System.out.
println("Divisor"); int S.nextInt(); System.out.println(a/b);}

```

Ainda considerando o conceito de *exception* e da sua influência na qualidade dos programas criados a partir da linguagem java, é importante abordar o recurso de *try...Catch*, cuja função capturar as exceções.

Diante disso, vamos conhecê-lo melhor e entender como funciona?

## Try...Catch

Neste momento, utilizaremos o *try...Catch* para capturar as exceções. No exemplo abaixo, executamos o programa para gerar o erro anterior, provocado por selecionarmos um número e o dividirmos por zero.

```

eclipse-workspace - meuprojeto/src/br/com/erros/DividePorZero.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
Package Explorer DividePorZero.java module-info.java
meuprojeto JRE System Library [JavaSE-15]
src br.com.erros DividePorZero.java br.com.heranca br.com.meuprojeto module-info.java
1 package br.com.erros;
2
3 import java.util.InputMismatchException;
4 import java.util.Scanner;
5
6 public class DividePorZero {
7
8     public static void main(String[] args) {
9
10         try {
11
12             Scanner s = new Scanner(System.in);
13             System.out.print("Número: ");
14             int a = s.nextInt();
15             System.out.print("Divisor : ");
16             int b = s.nextInt();
17
18             System.out.println(a / b);
19         }
20         catch(InputMismatchException e1) {
21             System.out.println("Erro de InputMismatchException capturado!");
22         }
23         catch(Throwable e2) {
24             System.out.println("Erro de ArithmeticException capturado!");
25         }
26     }
27 }
28

```

Console

```

<terminated> DividePorZero [Java Application] C:\Users\cmoreno\p2\pool\plugins\org.eclipse.jdt.openjdk.hotspot.jre.full.win32.x86
Número: 1
Divisor : 0
Erro de ArithmeticException capturado!

```

**#PraCegoVer:** na imagem, temos um *print* da tela do software Eclipse com *dashboard*, barra de ferramenta na parte superior, com o layout de pastas no canto esquerdo e com o painel de programação no lado direito, em que está escrito o seguinte código:

```

package br.com.erros; import java.util. Input MismatchException;
import java.util.Scanner; public class DividePorZero public static void
main(Strinell args) { try { Scanner s = new Scanner(System.in); System.
out.print("Número: "); int a.s.nextInt(); System.out.print("Divisor :"); int
bos.nextInt(); System.out.println(a/b); catch(Input MismatchException
e) { System.out.println("Erro de InputMismatchException capturado!");}
catch(Throwable e2) System.out.println("Erro de ArithmaticException
capturado!");}}

```

E se em vez de digitarmos um número, digitássemos um caractere qualquer? Qual seria o resultado? Consegue pensar em algo?

```

eclipse-workspace - meuprojeto/src/br/com/erros/DividePorZero.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
Package Explorer module-info.java DividePorZero.java
meuprojeto JRE System Library [JavaSE-15]
src
  br.com.erros
    DividePorZero.java
    br.com.heranca
    br.com.meuprojeto
    module-info.java
1 package br.com.erros;
2
3 import java.util.InputMismatchException;
4 import java.util.Scanner;
5
6 public class DividePorZero {
7
8     public static void main(String[] args) {
9
10         try {
11
12             Scanner s = new Scanner(System.in);
13             System.out.print("Número: ");
14             int a = s.nextInt();
15             System.out.print("Divisor : ");
16             int b = s.nextInt();
17
18             System.out.println(a / b);
19         }
20         catch(InputMismatchException e1) {
21             System.out.println("Erro de InputMismatchException capturado!");
22         }
23         catch(Throwable e2) {
24             System.out.println("Erro de ArithmeticException capturado!");
25         }
26     }
27 }
28

```

Console

```

<terminated> DividePorZero [Java Application] C:\Users\cmoreno\p2\pool\plugins\org.eclipse.jdt.openjdk.hotspot.jre.win32.x86_64
Número: XPTO
Erro de InputMismatchException capturado!

```

**#PraCegoVer:** na imagem, temos um *print* da tela do software Eclipse com *dashboard*, barra de ferramenta na parte superior, com o layout de pastas no canto esquerdo e com o painel de programação no lado direito, em que está escrito o seguinte código:

```

package br.com.erros; import java.util. Input MismatchException;
import java.util.Scanner; public class DividePorZero { public static void
main(String[] args) { try Scanner s = new Scanner(System.in); System.
out.print("Número: "); int a=s.nextInt(); System.out.print("Divisor : "); int
b. S.nextInt(); System.out.println(a/b); catch(Input MismatchException e) {
System.out.println("Erro de Input MismatchException capturado!"); catch(Throwable e2) {
System.out.println("Erro de ArithmeticException capturado!");}}}

```

Observe que com a troca, temos um erro de *InputMismatchException*. Dessa maneira, utilizamos o método *Throwable* para verificar os erros reproduzidos pelo programa.

Entretanto, devemos ressaltar que, independentemente da aplicação do *try...Catch*, é possível aplicar um bloco de códigos que pode operar de modo independente ao recurso anterior. Esse bloco é chamado de *finally*, siga em frente para conhecê-lo.

## Finally

O *finally* constitui em um bloco que será executado independentemente, apresentando ou não erros no método. Desse modo, na estrutura *try...Catch*, havendo ou não erro em qualquer ponto do bloco, o *finally* sempre será executado. Observe a sua aplicação no código:

```

eclipse-workspace - meuprojeto/src(br/com/erros/DividePorZero.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
Package Explorer module-info.java *DividePorZero.java
1 package br.com.erros;
2 import java.util.InputMismatchException;
3 import java.util.Scanner;
4
5 public class DividePorZero {
6
7     public static void main(String[] args) {
8
9         Scanner s = new Scanner(System.in);
10        boolean continuar = true;
11
12        do {
13            try {
14                System.out.print("Número: ");
15                int a = s.nextInt();
16                System.out.print("Divisor : ");
17                int b = s.nextInt();
18
19                System.out.println(a / b);
20                continuar = false;
21            } catch(InputMismatchException e1) {
22                System.out.println("Favor inserir números inteiros");
23                s.nextLine(); //descarta entrada inválida
24            } catch(Throwable e2) {
25                System.out.println("O divisor deve ser diferente de zero");
26            } finally {
27                System.out.println("Finally executado...");
28            }
29        } while (continuar);
30    }
31
32 }
33
34 }

```

Console

```

DividePorZero [Java Application] C:\Users\cmoreno\.p2\pool\plugins\org.eclipse.jdt.core\1.15.0.20150611_1400
Número: 1
Divisor : 0
O divisor deve ser diferente de zero
Finally executado...
Número:

```

**#PraCegoVer:** na imagem, temos um *print* da tela do software Eclipse com *dashboard*, barra de ferramenta na parte superior, com o layout de pastas no canto esquerdo e com o painel de programação no lado direito, em que está escrito o seguinte código:

```

package br.com.erros; import java.util. Input MismatchException;
import java.util.Scanner; public class DividePorZero { public static void
main(String[] args) { Scanner new Scanner(System.in); boolean continuar =
true; do try { System.out.print("Número: "); int a.s.nextInt(); System.
out.print("Divisor : "); int b s.nextInt(); System.out.println(a/b); continuar
false; catch(Input MismatchException el) System.out.println("Favor
inserir números inteiros"); S.nextLine();//descarta entrada inválida
catch(Throwable e2) { System.out.println("o divisor deve ser diferente de
zero"); finally System.out.println("Finally executado...");} while (continuar);}
}

```

Lembre-se que podemos alterar o atributo do *System.out* para *System.err*. Com essa mudança, o resultado obtido será destacado.

```

eclipse-workspace - meuprojeto/src(br.com.erros.DividePorZero.java) - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
Package Explorer module-info.java *DividePorZero.java
meuprojeto JRE System Library [JavaSE-15]
src br.com.erros DividePorZero.java
br.com.heranca br.com.meuprojeto module-info.java
1 package br.com.erros;
2 import java.util.InputMismatchException;
3 import java.util.Scanner;
4
5 public class DividePorZero {
6
7     public static void main(String[] args) {
8
9         Scanner s = new Scanner(System.in);
10        boolean continuar = true;
11
12        do {
13            try {
14                System.out.print("Número: ");
15                int a = s.nextInt();
16                System.out.print("Divisor : ");
17                int b = s.nextInt();
18
19                System.out.println(a / b);
20                continuar = false;
21            }
22            catch(InputMismatchException e1) {
23                System.err.println("Favor inserir números inteiros");
24                s.nextLine(); //descarta entrada inválida
25            }
26            catch(Throwable e2) {
27                System.err.println("O divisor deve ser diferente de zero");
28            }
29            finally {
30                System.out.println("Finally executado...");
31            }
32        } while (continuar);
33    }
34 }

```

Console

```

DividePorZero [Java Application] C:\Users\cmoreno.p2\pool\plugins\org.eclipse.jst.java.core\1.15.0
Número: 1
Divisor : 0
o divisor deve ser diferente de zero
Número: Finally executado...

```

**#PraCegoVer:** na imagem, temos um *print* da tela do software Eclipse com *dashboard*, barra de ferramenta na parte superior, com o layout de pastas no canto esquerdo e com o painel de programação no lado direito, em que está escrito o seguinte código:

```

package br.com.erros; import java.util. Input MismatchException;
import java.util.Scanner; public class DividePorZero { public static void
main(String[] args) { Scanner new Scanner(System.in); boolean continuar =
true; do try { System.out.print("Número: "); int a.s.nextInt(); System.
out.print("Divisor : "); int b s.nextInt(); System.out.println(a/b);
continuar false; catch(Input MismatchException el) System.out.println("Favor
inserir números inteiros"); S.nextLine();//descarta entrada inválida
catch(Throwable e2) { System.out.println("o divisor deve ser diferente de
zero"); finally System.out.println("Finally executado..."); while (continuar);}}
}

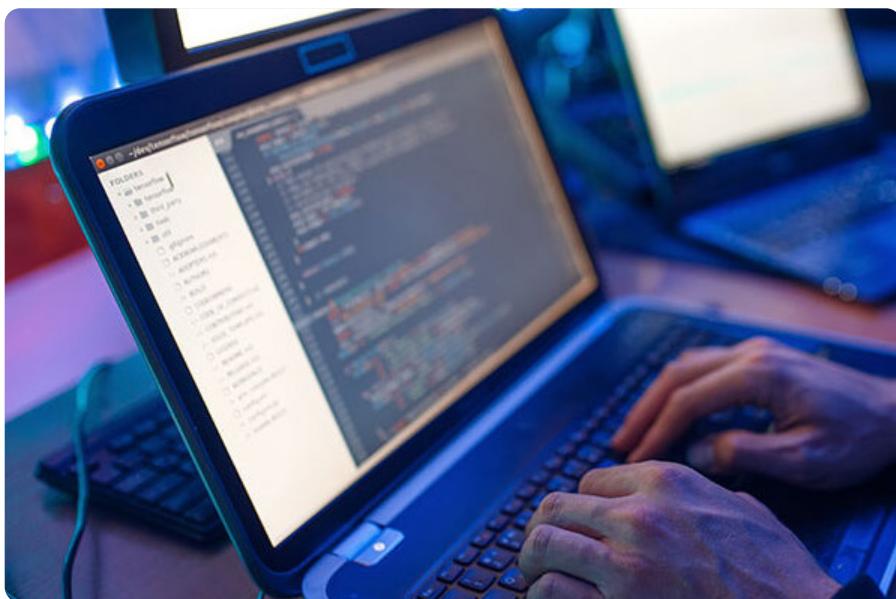
```

Agora que você já tem uma base de conhecimento sobre exceções e os recursos *try...Catch* e *finally*, observando como colocá-los em prática, podemos avançar para o próximo tópico, onde você estudará mais detalhadamente a respeito da interface gráfica.

Inclusive, o que você viu anteriormente será de grande valia, uma vez que as adequações das exceções e inconsistências nas linhas de código feitas pelos recursos *try...Catch* e *finally* vão otimizar e melhorar a experiência do usuário com o software a qual é mediada pela interface gráfica.

# Interface gráfica

O **Graphical User Interface** ou, simplesmente, GUI, diz respeito à interface gráfica com botões e menus, dentro de aplicações reais, em que o usuário pode interagir com as alternativas por meio de mouse e teclado.



**#PraCegoVer:** Na imagem, um programador está trabalhando em um notebook com a tela aberta, sendo que as suas mãos e o aparelho estão em primeiro plano. No fundo, atrás deles, existe um monitor de tela plana e, ao lado dela, outro notebook com a tela aberta e a imagem desfocada.

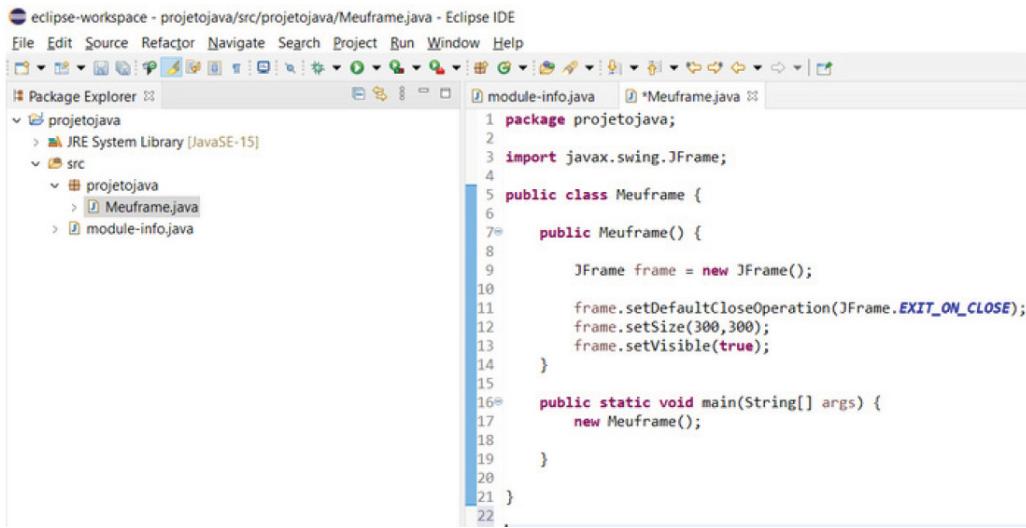
## Classes próprias

Existem algumas classes próprias para a construção das interfaces gráficas, as quais são inseridas no pacote “javax.swing”, no pacote “java.awt” e em outras bibliotecas de mercado.

## Aplicações Android

Quando construímos aplicações para Android, por exemplo, é disponibilizado uma API em Java com as bibliotecas de interface gráfica.

Assim, a interface gráfica começa de uma janela (*JFrame*). Para tanto, você precisa criar a primeira janela. Vamos colocar a ideia em prática? Acompanhe os dois primeiros passos a seguir!



The screenshot shows the Eclipse IDE interface. The top menu bar includes File, Edit, Source, Refactor, Navigate, Search, Project, Run, Window, and Help. Below the menu is a toolbar with various icons. On the left is the Package Explorer view, which shows a project named 'projetojava' containing a package 'projetojava' with a class 'Meuframe.java'. The right side of the interface is the code editor, displaying the following Java code:

```

eclipse-workspace - projetojava/src/projetojava/Meuframe.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
Package Explorer module-info.java *Meuframe.java
projetojava JRE System Library [JavaSE-15]
src projetojava
Meuframe.java module-info.java
1 package projetojava;
2
3 import javax.swing.JFrame;
4
5 public class Meuframe {
6
7     public Meuframe() {
8
9         JFrame frame = new JFrame();
10
11         frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
12         frame.setSize(300,300);
13         frame.setVisible(true);
14     }
15
16     public static void main(String[] args) {
17         new Meuframe();
18     }
19 }
20
21 }
22 .

```

**#PraCegoVer:** na imagem, temos um *print* da tela do software Eclipse com *dashboard*, barra de ferramenta na parte superior, com o layout de pastas no canto esquerdo e com o painel de programação no lado direito, em que está escrito o seguinte código:

```
package projetojava; import javax.swing.JFrame; public class Meuframe { public Meuframe() { JFrame frame = new JFrame(); frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE); frame.setSize(300, 300); frame.setVisible(true); public static void main(String[] args) { new Meuframe();}}}
```

Inicialmente, crie um projeto chamado “projetojava”. Depois, é preciso criar um pacote com o nome “projetojava” e uma classe chamada *Meuframe*.

```
public Meuframe() {
    JFrame frame = new JFrame();
```

Iniciamos no método construtor o *frame*, representado por *JFrame*.

Um *frame* pode conter dimensões, títulos, menus e botões, bem como executar ações de abrir ou fechar. De modo geral, podemos atribuir todos esses atributos ao *frame*. Temos, então, as seguintes definições:

**`frame.setSize (300,300)`**

Define o tamanho do frame (300 *pixels* de largura e 300 *pixels* de altura).

**`frame.setVisible (true)`**

Define o *frame* como visível (pode estar oculto).

**`frame.setDefaultCloseOperation (JFrame.EXIT_ON_CLOSE)`**

Permite ao usuário fechar o *frame* assim que clicar no botão “X”.

Veja como as definições são aplicadas na prática, com o exemplo do código a seguir.

```
frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
frame.setSize(300,300);
frame.setVisible(true);
```

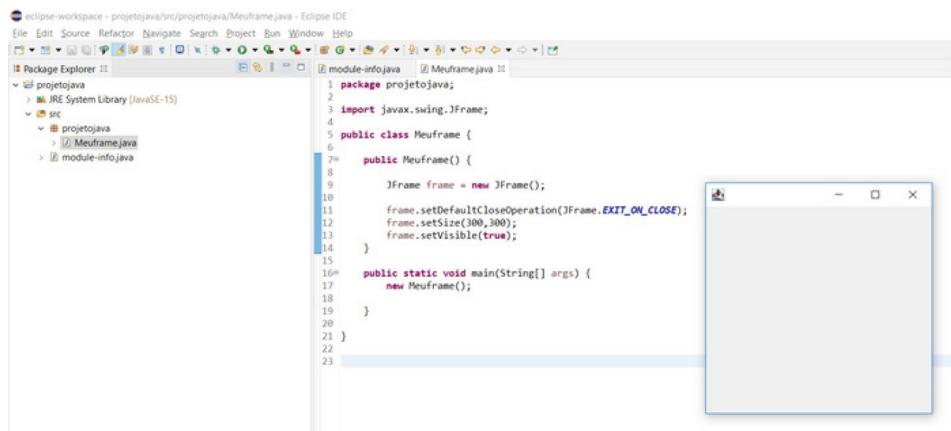
**#PraCegoVer:** na imagem, temos um código com as definições do *frame*. O código é: `frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE); frame.setSize(300, 300); frame.setVisible(true);`

Não devemos esquecer, ainda, do “xxx”, que permite o uso do *JFrame*.

```
package projetojava;
```

**#PraCegoVer:** na imagem, temos uma linha de código indicando o uso de *JFrame*. O código é: `package projetojava;`

Agora que você viu os principais pontos do programa, já é possível executá-lo e perceber o resultado da criação do *frame*. Dessa maneira, você conseguirá realizar ações simples, como minimizar, maximizar, fechar e redimensionar.



**#PraCegoVer:** na imagem, temos um *print* da tela do software Eclipse com *dashboard*, barra de ferramenta na parte superior, com o layout de pastas no canto esquerdo e com o painel de programação no lado direito, em que está escrito o seguinte código:

```
package projetojava; import javax.swing.JFrame; public class Meuframe { public Meuframe() { JFrame frame = new JFrame(); frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE); frame.setSize(300, 300); frame.setVisible(true); } public static void main(String[] args) { new Meuframe(); }}
```

Ao lado do código, é aberto um pop up.

Prosseguiremos, então, com a implantação do código. A partir do método apresentado na próxima imagem, será possível recuperar o contêiner do *frame* e adicionar um controle. Para um melhor entendimento, siga os passos descritos no código, para adicionar esse controle:

```
package projetojava;

import javax.swing.JFrame;
public class Meuframe {
    public Meuframe() {

        JFrame frame = new JFrame();
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setSize(300, 300);
        frame.setVisible(true);
    }
    public static void main(String[] args) {
        new Meuframe();
    }
}
```

### Passo 1

Definimos, primeiro, o tipo de controle. Neste caso, teremos um botão.

### Passo 2

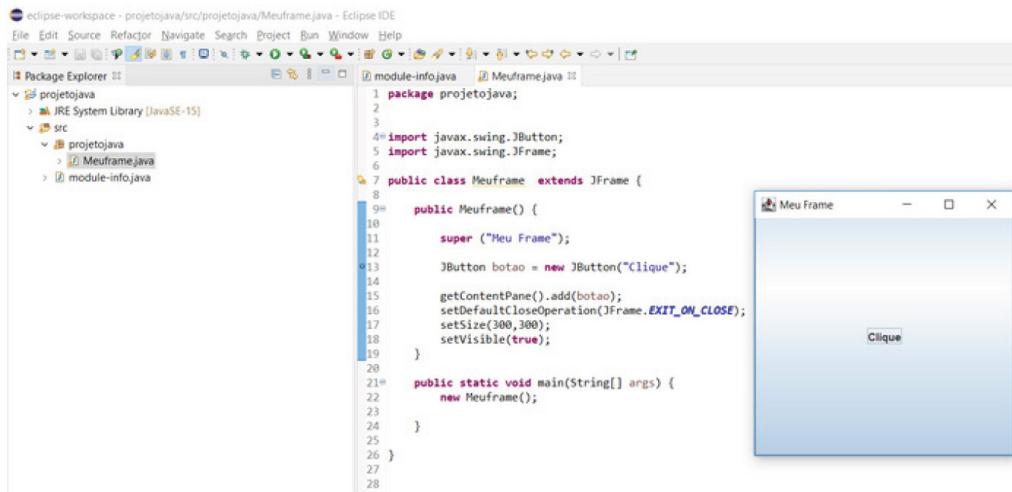
Para adicionar o botão que desejamos, utilizamos a classe *JButton()*.

### Passo 3

Depois, estendemos a classe *JFrame*, pois, no método construtor, não precisamos utilizar o método *JFrame* por ser um *frame*.

Após realizar as ações, você precisa configurar um título para o *frame*. Entretanto, é de suma importância observar que, para o código funcionar, será necessário

adicionar um XPTO. Na sequência, executa-se o programa e observe o resultado.



**#PraCegoVer:** na imagem, temos um *print* da tela do software Eclipse com *dashboard*, barra de ferramenta na parte superior, com o layout de pastas no canto esquerdo e com o painel de programação no lado direito, em que está escrito o seguinte código:

```

package projetojava; import javax.swing.JFrame; public class
Meuframe { public Meuframe() { JFrame frame = new JFrame(); frame.
setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE); frame.setSize(300,
300); frame.setVisible(true); public static void main(String[]
args) { new Meuframe();}}

```

Ao lado do código, é aberto um pop up denominado "Meu Frame" no qual está escrito no centro "clique".

Com essa introdução a respeito da interface gráfica e como ela funciona na prática, chegou a hora de conhecer os elementos do *frame* e como eles influenciam no resultado do seu programa. Acompanhe!

## GUI layout, border e flow layout

A disposição dos elementos no *frame*, depende do layout que você adotará na janela. Por conta disso, a partir de agora, vamos entender como funcionam os layouts. Assista ao vídeo a seguir para uma explicação mais detalhada!



## Vídeo

Confira o [vídeo](#) de como criar um layout.

Perdeu algum detalhe? Confira o que foi abordado no vídeo.

Olá! Vamos criar um layout e verificar como ficam os elementos do frame?

O primeiro passo é criar uma classe chamada Layouts utilizando o método “main”. Em seguida, criar o código e executá-lo.

```
package projetojava;

import javax.swing.JButton;
import javax.swing.JFrame;

public class Layouts extends JFrame {

    public Layouts() {
        super("Meu layout");

        getContentPane().add(new JButton("botão centralizado"));
        getContentPane().add(new JButton("botão centralizado 2"));

        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setSize(300, 300);
        setVisible(true);
    }

    public static void main(String[] args{
        new Layouts();
    }
}
```

Ao inserir uma nova linha “adicionando” um botão, será exibida a segunda linha como resultado, pois esse é o layout padrão desse contêiner.

Vale ressaltar que todos contêineres aceitam layouts e controles.

Dessa forma, no próprio layout, é definido de que modo cada controle e contêiner serão incluídos no local desejado. Confira.

```
package projetojava;

import javax.swing.JButton;
import javax.swing.JFrame;

public class Layouts extends JFrame {

    public Layouts() {
        super("Meu layout");

        getContentPane().add(new JButton("botão centralizado"));
        getContentPane().add(new JButton("botão centralizado 2"));
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        setSize(300,300);
        setVisible(true);
    }

    public static void main(String[] args) {
        new Layouts();
    }
}
```

O formato padrão utilizado pelo JFrame é o BorderLayout. Nele, são definidas algumas posições, nas quais podemos inserir os controles.

Dessa forma, no próprio layout, é definido de que modo cada controle e contêiner serão incluídos no local desejado. Confira.

```
package projetojava;

import javax.swing.JButton;
import javax.swing.JFrame;

public class Layouts extends JFrame {

    public Layouts() {
        super("Meu layout");

        getContentPane().add(new JButton("botão centralizado"));
        getContentPane().add(new JButton("botão centralizado 2"));
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        setSize(300,300);
        setVisible(true);
    }

    public static void main(String[] args) {
        new Layouts();
    }
}
```

O formato padrão utilizado pelo JFrame é o BorderLayout. Nele, são definidas algumas posições, nas quais podemos inserir os controles.

Veja como fica o programa.

```
package projetojava;

import javax.swing.JButton;
import javax.swing.JFrame;

import java.awt.BorderLayout;
import java.awt.Container;

public class Layouts extends JFrame {

    public Layouts() {
        super ("Meu Layout");

        Container c = getContentPane();

        c.add(BorderLayout.NORTH, new JButton("Botão Norte"));
        c.add(BorderLayout.SOUTH, new JButton("Botão Sul"));

        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setSize(300,300);
        setVisible(true);

    }

    public static void main(String[] args) {
        new Layouts();

    }

}
```

Agora, confira como fica o programa completo:

```
package projetojava;

import javax.swing.JButton;
import javax.swing.JFrame;

import java.awt.BorderLayout;
import java.awt.Container;

public class Layouts extends JFrame {

    public Layouts() {
        super ("Meu Layout");

        Container c = getContentPane();

        c.add(BorderLayout.NORTH, new JButton("Botão Norte"));
        c.add(BorderLayout.SOUTH, new JButton("Botão Sul"));
        c.add(BorderLayout.CENTER, new JButton("Botão Centro"));
        c.add(BorderLayout.EAST, new JButton("Botão Leste"));
        c.add(BorderLayout.WEST, new JButton("Botão Oeste"));

        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setSize(300,300);
        setVisible(true);
    }

    public static void main(String[] args) {
        new Layouts();
    }
}
```

Mas, não se esqueça, para que os botões executem algo, é necessário tratar os eventos desejados.

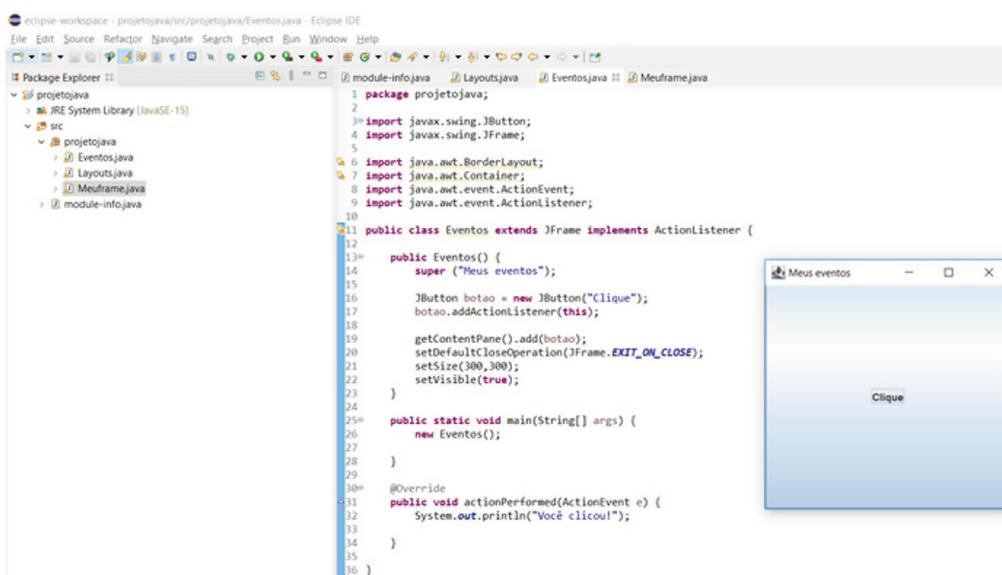
Assunto que você acompanha com mais detalhes no próximo tópico.

Com o vídeo você entendeu a importância que os recursos estudados têm, sobretudo, em relação à construção da interface gráfica dos softwares. Neste tópico vimos como inserir botões no layout, lembrando que existem muitos layouts, mas eles não realizam qualquer ação.

Para que os botões executem algo, de fato, devemos tratar os seus eventos. Assim, no próximo tópico, você entenderá sobre esse conceito e verá a maneira pela qual é possível aplicá-lo usando a linguagem Java.

## Eventos

Os eventos ocorrem quando interagimos com os componentes da janela, como quando passamos o mouse em cima de um botão e a cor se destaca ou quando clicamos em um desses botões. Eles são os responsáveis pela geração dos eventos. Acompanhe um exemplo prático!



**#PraCegoVer:** na imagem, temos um *print* da tela do software Eclipse com *dashboard*, barra de ferramenta na parte superior, com o layout de pastas no canto esquerdo e com o painel de programação no lado direito, em que está escrito o seguinte código:

```

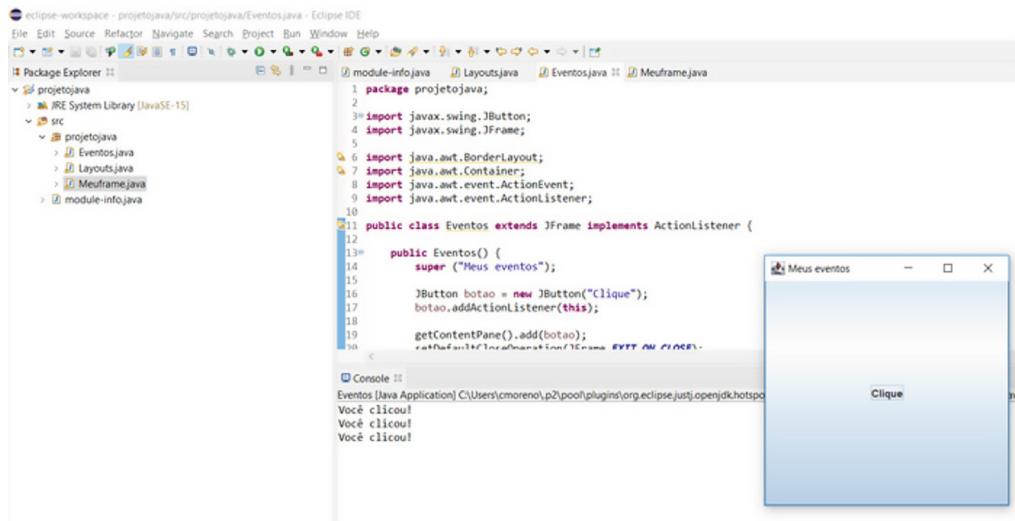
package projetojava; import javax.swing. Button: import javax.swing.
JFrame; import java.awt.BorderLayout: import java.awt.Container: import
java.awt.event, ActionEvent: import java.awt.event. ActionListener;
public class Eventos extends JFrame implements ActionListener {
    public Eventos super (Meus eventos; Meus eventos - O X Button botao
    - new JButton("Clique"); botao.addActionListener(this); getContentPane().
    add(botao); setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    setSize(300,300); setVisible(true); Clique public static void main(String[args)

```

```
new Eventos(); @Override public void actionPerformed(ActionEvent e)
System.out.println("Você clicou!");}
```

Ao lado do código, é aberto um pop up denominado "Meu Evento" no qual está escrito no centro "Clique".

Criamos uma classe chamada Eventos, selecionando o método "*main*". Dessa maneira, o programa ficará conforme apresentado aqui. Em seguida, pode executar o programa e observar o resultado.



**#PraCegoVer:** na imagem, temos um *print* da tela do software Eclipse com *dashboard*, barra de ferramenta na parte superior, com o layout de pastas no canto esquerdo e com o painel de programação no lado direito, em que está escrito o seguinte código:

**#PraCegoVer:** na imagem, temos um print da tela do software Eclipse com dashboard, barra de ferramenta na parte superior, com o layout de pastas no canto esquerdo e com o painel de programação no lado direito, em que está escrito o seguinte código: package projetojava; import javax.swing.JFrame; public class Meuframe { public Meuframe() { JFrame frame = new JFrame(); frame.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE); frame.setSize(300, 300); frame.setVisible(true); } public static void main(String[] args) { new Meuframe(); }}

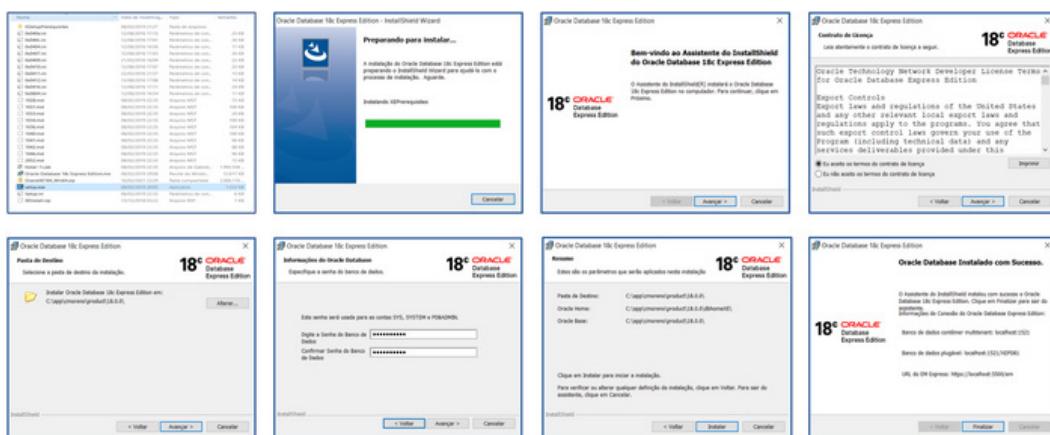
Ao lado do código, é aberto um pop up denominado "Meu Frame" no qual está escrito no centro "Botão Centralizado".

No final, ao clicar no botão, observe que será exibida a informação implementada.

Viu como é simples o procedimento de para criar eventos a partir do recurso botão expresso anteriormente?

Existe um outro elemento relevante para o seu futuro profissional enquanto desenvolvedor que emprega a linguagem Java, que é a integração com banco de dados *Oracle Express Edition*. Tema do próximo tópico.

Quando finalizar o download, navegue até o diretório para extrair o pacote e executar o setup de instalação. O processo é simples e intuitivo, basta seguir o passo a passo.



Após concluir a instalação, vá até o menu “Iniciar”, digite “sqlplus” e selecione a opção “Executar”. Observe que ele pede o nome do usuário.

```
SQL*Plus
```

SQL\*Plus: Release 18.0.0.0.0 - Production on Qua Fev 17 12:34:47 2021  
Version 18.4.0.0.0  
  
Copyright (c) 1982, 2018, Oracle. All rights reserved.  
  
Informe o nome do usuário:

Para acessar, então, digite o usuário **system**. Este possui acesso irrestrito ao banco de dados. Você também deve informar a senha criada no momento da instalação do Oracle XE.

```
C:\app\cmoreno\product\18.0.0\dbhomeXE\bin\sqlplus.exe
```

SQL\*Plus: Release 18.0.0.0.0 - Production on Qua Fev 17 12:40:25 2021  
Version 18.4.0.0.0  
  
Copyright (c) 1982, 2018, Oracle. All rights reserved.  
  
Informe o nome do usuário: system  
Informe a senha:  
Horário do log-in bem-sucedido: Qua Fev 17 2021 12:40:15 -03:00  
  
Conectado a:  
Oracle Database 18c Express Edition Release 18.0.0.0.0 - Production  
Version 18.4.0.0.0  
  
SQL>

Execute o comando demonstrado para que seja possível a criação do usuário, a fim de acessar o banco de dados.

```
Conectado a:  
Oracle Database 21c Express Edition Release 21.0.0.0 - Production  
Version 21.0.0.0  
  
SQL > alter session set "_ORACLE_SCRIPT"= true;  
  
Sessão alterada.
```

Com isso, será possível executar comandos no banco de dados. Logo em seguida, a primeira tarefa a ser realizada é criar um usuário para utilizar na atividade. Assim, digite o comando conforme indicado.

```
SQL> create user curso identified by curso1234;  
  
Usuário criado.
```

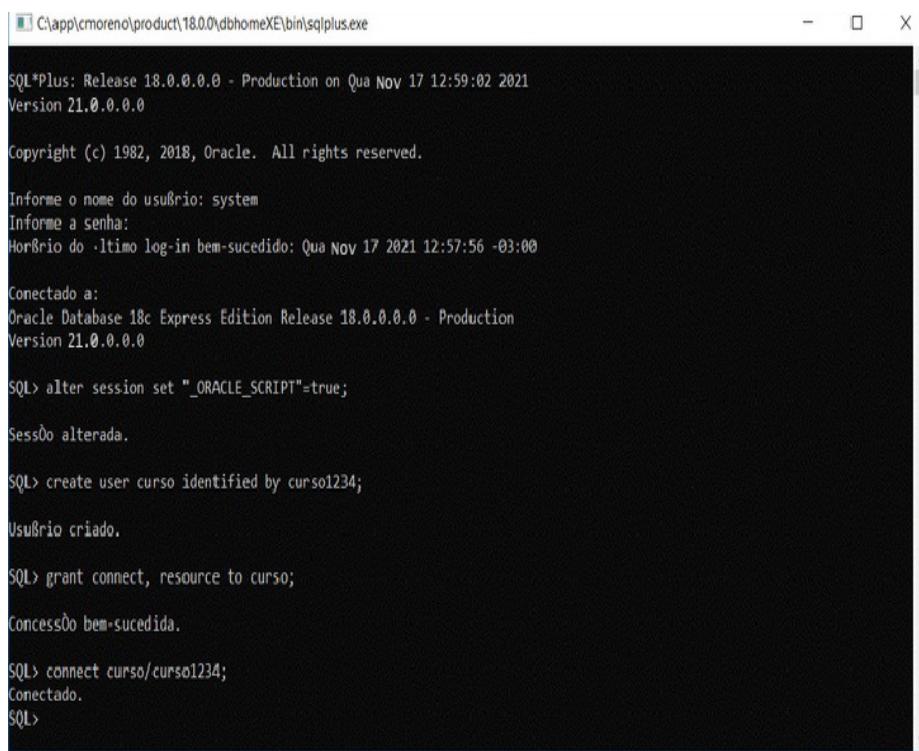
Agora, é preciso atribuir ao usuário alguns privilégios para que ele possa se conectar ao banco de dados e ter acesso às tabelas, a fim de incluir ou excluir os dados.

```
Usuário criado.  
  
SQL> grant connect, resource to curso;  
  
Concessão bem-sucedida.
```

Em seguida, conecte ao banco de dados utilizando o usuário criado.

```
SQL> connect curso/curso1234;  
Conectado.  
SQL>
```

Ao final, teremos algo como o observado nesta imagem.



```
C:\app\cmoreno\product\18.0.0\dbhomeXE\bin\sqlplus.exe

SQL*Plus: Release 18.0.0.0 - Production on Qua Nov 17 12:59:02 2021
Version 21.0.0.0

Copyright (c) 1982, 2018, Oracle. All rights reserved.

Informe o nome do usuário: system
Informe a senha:
Horário do último log-in bem-sucedido: Qua Nov 17 2021 12:57:56 -03:00

Conectado a:
Oracle Database 18c Express Edition Release 18.0.0.0 - Production
Version 21.0.0.0

SQL> alter session set "_ORACLE_SCRIPT=true";
Sessão alterada.

SQL> create user curso identified by curso1234;
Usuário criado.

SQL> grant connect, resource to curso;
Concessão bem-sucedida.

SQL> connect curso/curso1234;
Conectado.
SQL>
```

Como o banco de dados será utilizado para estudos, concederemos o **privilegio de DBA** ao usuário “curso”. Assim, o acesso será irrestrito.

```
SQL> GRANT DBA TO CURSO;
Concessão bem-sucedida.

SQL> COMMIT;
Commit concluído.
```

Aqui no curso, lembramos que nosso objetivo é fazer você entender os conceitos básicos e necessários para executar o projeto Java, e não ensinar sobre o banco de dados.

Assim, como já estamos conectados com o usuário, será o momento de criar uma tabela. Desse modo, crie a tabela “PESSOA”, a qual deverá apresentar as seguintes informações: Código, nome e sexo.

PESSOA	
◆ CODIGO (PK)	NUMBER(2,0)
◆ NOME	VARCHAR2(30)
◆ SEXO	CHAR(1)

Veja como fica o SQL.

```
SQL> CREATE TABLE PESSOA (
  2  CODIGO      NUMBER(2,0)      NOT NULL,
  3  NOME        VARCHAR2(15),
  4  SEXO        CHAR(1),
  5  EMAIL        VARCHAR2(30),
  6  CONSTRAINT PK1      PRIMARY KEY(CODIGO)
  7 );
```

Tabela criada.

Depois de observar a estrutura da tabela “PESSOA”, podemos perceber o seguinte em “DESC PESSOA”:

```
SQL> DESC PESSOA;
Nome                      Nulo?    Tipo
-----
CODIGO                    NOT NULL NUMBER(2)
NOME                     VARCHAR2(30)
SEXO                      CHAR(1)

SQL>
```

Para exibir a quantidade de registros na tabela, realize o comando: “SQL>SELECT COUNT(\*) FROM PESSOA” e verifique que, nela, ainda não há registros.

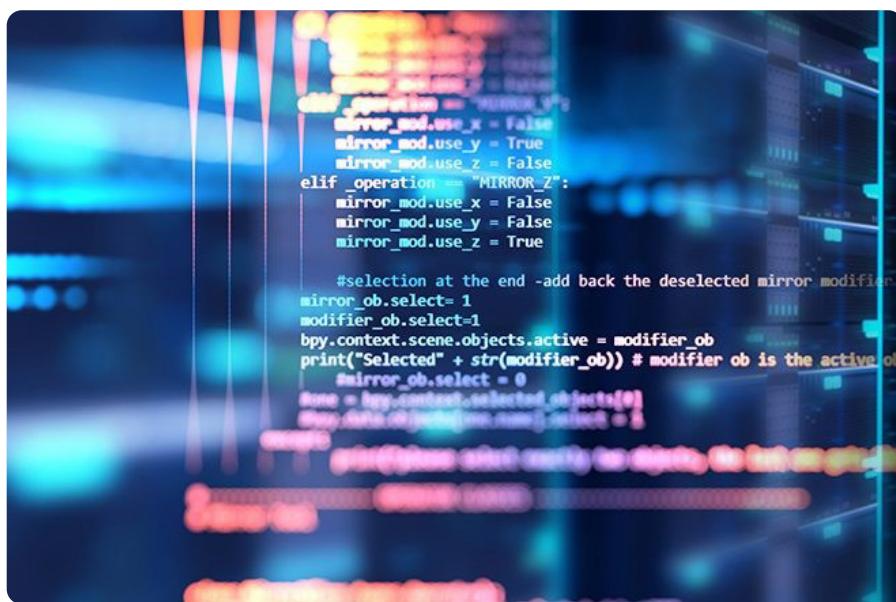
```
SQL> SELECT COUNT(*) FROM PESSOA;

COUNT(*)
-----
0

SQL>
```

# Integração com banco de dados *Oracle Express Edition*

O *Java Database Connectivity* (JDBC) diz respeito à tecnologia utilizada pelos programas Java para se comunicarem com os bancos de dados.



**#PraCegoVer:** Na imagem, há uma série de servidores verticais enfileirados ao fundo com a imagem desfocada. Em primeiro plano, existem uma série de linhas de código em destaque.

## Armazenamento de dados

Sabemos que não existem sistemas corporativos sem bancos de dados, não é mesmo? A maioria das empresas armazena seus dados em diversos arquivos, como em folhas de pagamento, contas a pagar, contas a receber, inventário etc.

## Sistema de Gerenciamento de Banco de Dados (SGBD)

Os sistemas criados para gerenciar os dados ou as informações são chamados de **Sistemas de Gerenciamento de Bancos de Dados (SGBD)**. Os bancos de dados populares consistem naqueles de dados relacionais, sendo que a linguagem utilizada é

o *Structured Query Language*, conhecida como SQL.

Um banco de dados relacional armazena dados em tabelas, as quais são compostas de linhas. Estas são distribuídas em colunas, armazenando os dados.

Veja na prática, de modo simples, como instalar o *Oracle Express Edition*, bem como criar e popular a base de dados com informações. Realize a configuração e conexão do programa utilizando o *driver JDBC*, finalizando com o desenvolvimento do programa conectando à base de dados. Vamos começar?

## Instalando o *Oracle Express Edition*

Para instalar o Oracle Express Edition, acesse o site para download da versão Windows x64. Feito isso, acompanhe o vídeo que preparamos para você com o passo a passo!

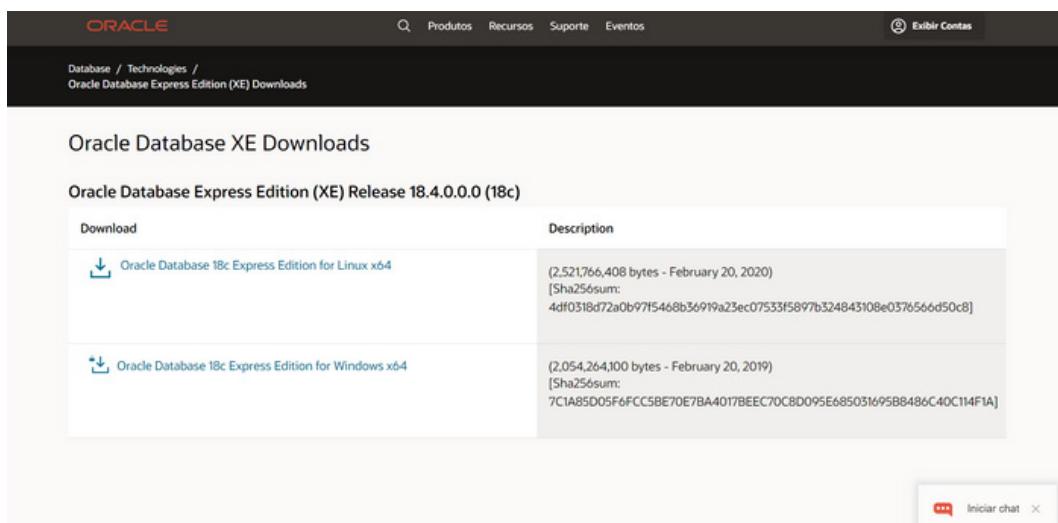


### Vídeo

Confira o [vídeo](#) sobre Instalando o *Oracle Express Edition*.

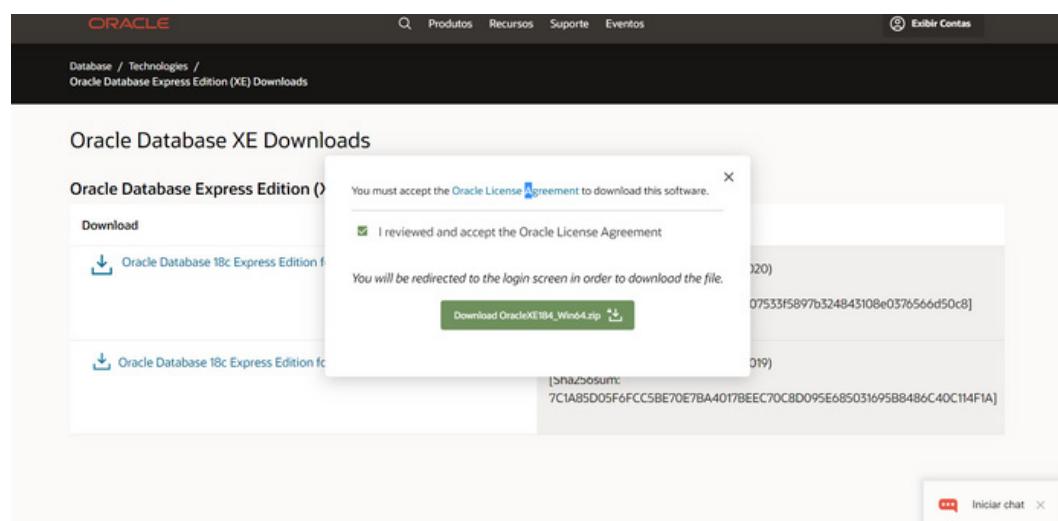
Perdeu algum detalhe? Confira o que foi abordado no vídeo.

No [site](#), faça o download da versão Windows selecionando a opção o “Oracle Database 18c Express Edition for Windows x64”.



The screenshot shows the Oracle Database XE Downloads page. At the top, there's a navigation bar with links for Oracle, Produtos, Recursos, Suporte, Eventos, and Exibir Contas. Below the navigation, the URL is Database / Technologies / Oracle Database Express Edition (XE) Downloads. The main content area is titled "Oracle Database XE Downloads" and features a table with two rows. The first row is for "Oracle Database 18c Express Edition for Linux x64" and the second for "Oracle Database 18c Express Edition for Windows x64". Both rows have a "Download" link and a "Description" column. The "Description" column for the Linux version states "(2,521,766,408 bytes - February 20, 2020) [Sha256sum: 4dff0318d72a0b97f5468b36919a23ec07533f5897b324843108e0376566d50c8]". The "Description" column for the Windows version states "(2,054,264,100 bytes - February 20, 2019) [Sha256sum: 7C1A85D05F6FCC5BE70E7BA4017BEEC70C8D095E685031695B8486C40C114F1A]". At the bottom right of the page is a "Iniciar chat" button.

Lembrando que é necessário ter uma conta no site da Oracle, que pode ser criada gratuitamente, para realizar o download do Oracle XE.



The screenshot shows the same Oracle Database XE Downloads page as the previous one, but with a modal dialog box overlaid. The dialog box has a header "You must accept the Oracle License Agreement to download this software." and a checkbox labeled "I reviewed and accept the Oracle License Agreement". Below the checkbox, it says "You will be redirected to the login screen in order to download the file." A "Download OracleXE184\_Win64.zip" button is visible. The background of the page shows the two download options for Linux and Windows. At the bottom right of the page is a "Iniciar chat" button.

O próximo passo será inserir 10 registros na tabela em questão,

iniciando em `INSERT INTO PESSOA VALUES`

Seguido de:

- (1, 'CARLOS', 'M');
- (2, 'DANIEL', 'M');
- (3, 'JEFFERSON', 'M');
- (4, 'LEONARDO', 'M');
- (5, 'LUCCAS', 'M');
- (6, 'LIDIANA', 'F');
- (7, 'ALESSANDRA', 'F');
- (8, 'LIVIA', 'F');
- (9, 'ERICA', 'F');
- (10, 'PATRICIA', 'F').

Finalize a ação com o comando “`commit;`”.

```
SQL> INSERT INTO PESSOA VALUES (1, 'CARLOS', 'M');
1 linha criada.

SQL> INSERT INTO PESSOA VALUES (2, 'DANIEL', 'M');
1 linha criada.

SQL> INSERT INTO PESSOA VALUES (3, 'JEFFERSON', 'M');
1 linha criada.

SQL> INSERT INTO PESSOA VALUES (4, 'LEONARDO', 'M');
1 linha criada.

SQL> INSERT INTO PESSOA VALUES (5, 'LUCCAS', 'M');
1 linha criada.

SQL> INSERT INTO PESSOA VALUES (6, 'LIDIANA', 'F');
1 linha criada.

SQL> INSERT INTO PESSOA VALUES (7, 'ALESSANDRA', 'F');
1 linha criada.

SQL> INSERT INTO PESSOA VALUES (8, 'LIVIA', 'F');
1 linha criada.

SQL> INSERT INTO PESSOA VALUES (9, 'ERICA', 'F');
1 linha criada.

SQL> INSERT INTO PESSOA VALUES (10, 'PATRICIA', 'M');
1 linha criada.

SQL> commit;
Commit concluído.
```

Agora, realize a contagem da quantidade de registros e faça um “select” para retornar os registros criados no banco de dados. Pronto!

```
SQL> SELECT COUNT(*) FROM PESSOA;

COUNT(*)
-----
10

SQL> SELECT * FROM PESSOA;

CÓDIGO NOME      S
-----
1 CARLOS        M
2 DANIEL        M
3 JEFFERSON     M
4 LEONARDO      M
5 LUCCAS        M
6 LIDIANA        F
7 ALESSANDRA    F
8 LIVIA          F
9 ERICA          F
```

Com o passo a passo da instalação e um exemplo prático da criação de uma tabela, as coisas ficam mais fáceis, não é mesmo? Agora faça a sua instalação e inicie seus trabalhos!

De acordo com as instruções passadas no vídeo, com o programa *Oracle Express Edition* devidamente instalado, vamos aprender como utilizar o driver JDBC para conectar com a aplicação Java. Veja sobre isso no próximo tópico. Confira o conteúdo com atenção!

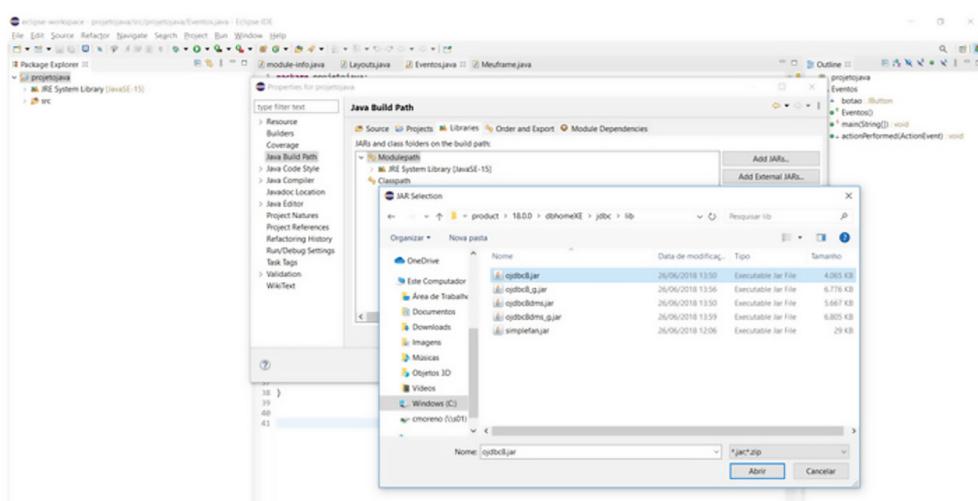
## Utilizando o *driver JDBC* para conectar o banco de dados

Para que seja possível conectar o banco de dados por meio das classes Java, utilizamos um *driver JDBC*. Ele se encontra na instalação do Oracle XE no seguinte diretório: “C:\app\product\18.0.0\dbhomeXE\jdbc\lib”.

ojdbc8.jar	26/06/2018 13:50	Executable Jar File	4.065 KB
ojdbc8_g.jar	26/06/2018 13:56	Executable Jar File	6.776 KB
ojdbc8dms.jar	26/06/2018 13:50	Executable Jar File	5.667 KB
ojdbc8dms_g.jar	26/06/2018 13:59	Executable Jar File	6.805 KB
simplefan.jar	26/06/2018 12:06	Executable Jar File	29 KB

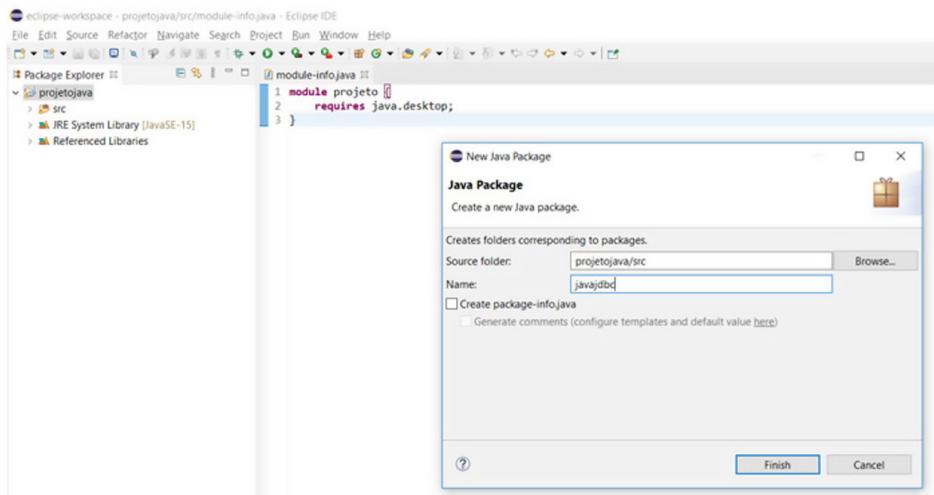
#PraCegoVer: na imagem, temos um *print* indicando o *driver* “ojdbc8.jar”.

Para importar o *driver JDBC* para o seu projeto Java, selecione com o botão direito do mouse no projeto “projetojava”. Navegue até “propriedades” para chegar na opção “Java Build Path”. Na guia “Libraries” selecione o botão “Add External JARs” e procure pelo *driver JDBC*. Será necessário selecionar o *driver* “ojdbc8.jar”, clicar no botão “Abrir” e, em seguida, em “Apply and Close”.



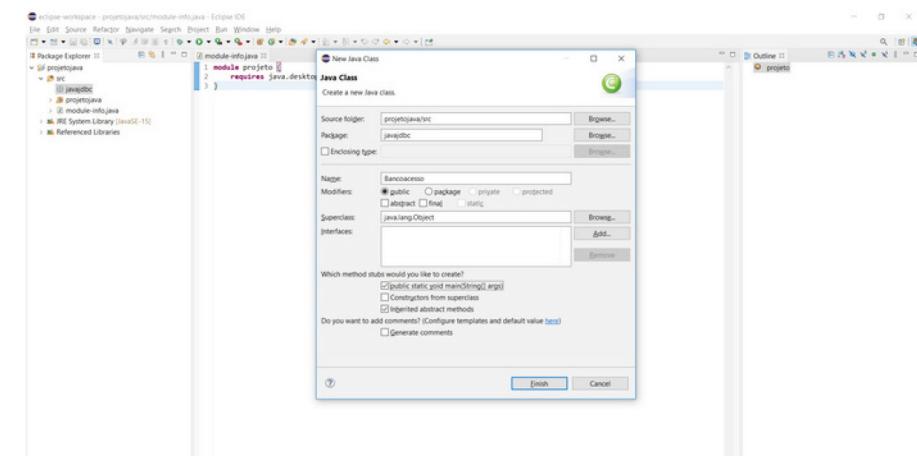
#PraCegoVer: na imagem, temos um *print* indicando o caminho para encontrar o *driver* “ojdbc8.jar” explicado no texto.

Após esse processo, crie um pacote chamado “javajdbc”.

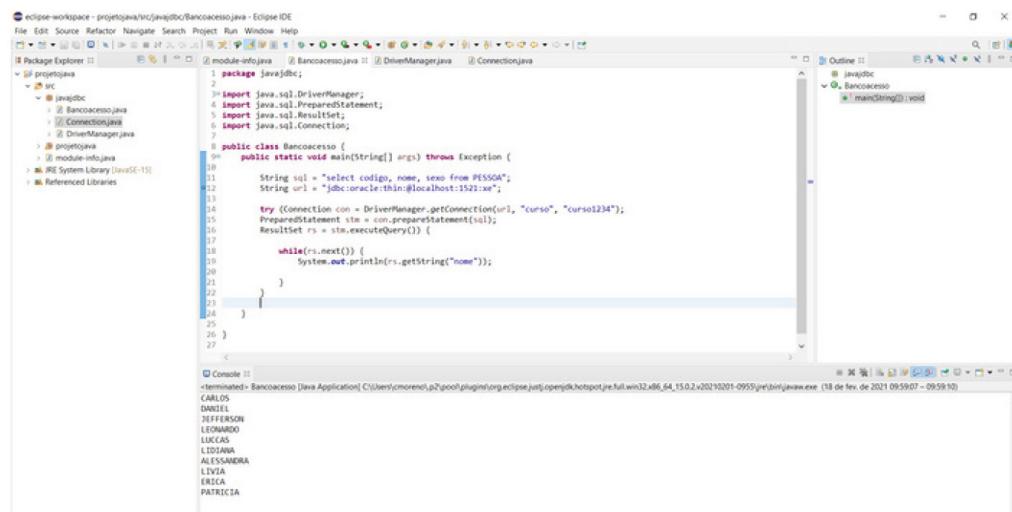


#PraCegoVer: na imagem, temos um *print* indicando o processo de criação do pacote “javajdbc”.

No pacote, siga mais um passo a passo com atenção. Acompanhe!



Crie uma classe chamada Bancoacesso e, na sequência, ative o método “main”.



The screenshot shows the Eclipse IDE interface with the following details:

- Project Explorer:** Shows the project structure with a package named `javajdbc` containing files `Bancoacesso.java`, `Connection.java`, and `DriverManager.java`.
- Code Editor:** Displays the `Bancoacesso.java` file with the following code:

```

package javajdbc;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.Connection;

public class Bancoacesso {
    public static void main(String[] args) throws Exception {
        String sql = "select codigo, nome, sexo from PESSOA";
        String url = "jdbc:oracle:thin:@localhost:1521:xe";
        try (Connection con = DriverManager.getConnection(url, "curso", "curso1234");
             PreparedStatement stm = con.prepareStatement(sql);
             ResultSet rs = stm.executeQuery()) {
            while(rs.next()) {
                System.out.println(rs.getString("nome"));
            }
        }
    }
}

```
- Console:** Shows the output of the program, listing names from a database table:

```

CARLOS
DANIEL
JEFFERSON
LEONARDO
LUCAS
LIDIANA
ALEXANDRA
LÉVIA
ERICA
PATRÍCIA

```

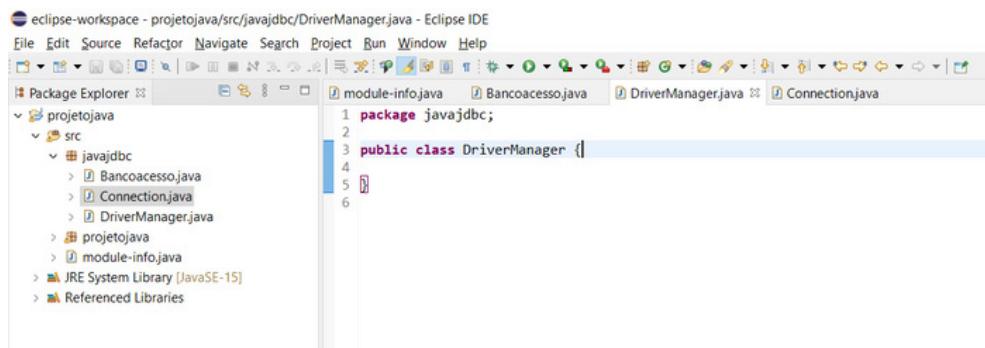
**#PraCegoVer:** na imagem, temos um *print* da tela do *software Eclipse* com *dashboard*, barra de ferramenta na parte superior, com o layout de pastas no canto esquerdo e com o painel de programação no lado direito, em que está escrito o seguinte código:

```

package javajdbc; import java.sql DriverManager; import java.sql.
PreparedStatement; import java.sql.ResultSet; import java.sql.
Connection; public class Bancoacesso {public static void main {String{}}
args} throwg Exception{ String sql = "select código, nome, sexo
from PESSOA"; String url = "jdbc : oracle:thin:@localhost:1521: exe";
Try {Connection con = DriverManager.getConnection) url, " curso",
"curso1234"; }PreparedStatement }stm }= }con.prepareStatement
(sql); ResultSet rs = stm.executeQuery()} { While (rs.next()){ System.out.
println(rs.getString("nome"));}}}

```

Digite o programa conforme indicado na imagem. Dessa maneira, realize a conexão com o banco de dados, requisitando a exibição de todos os registros do campo “NOME” da tabela “PESSOA”.



The screenshot shows the Eclipse IDE interface with the following details:

- Project Explorer:** Shows the project structure with a package named `javajdbc` containing files `Bancoacesso.java`, `Connection.java`, and `DriverManager.java`.
- Code Editor:** Displays the `DriverManager.java` file with the following code:

```

package javajdbc;
public class DriverManager {}
```

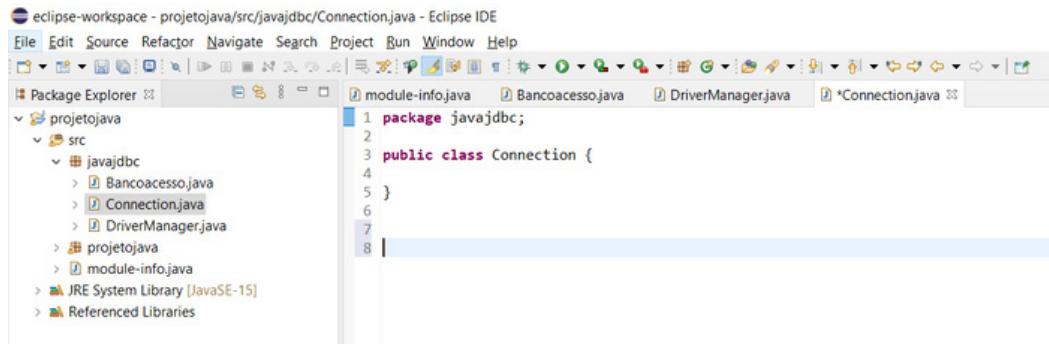
**#PraCegoVer:** na imagem, temos um *print* da tela do *software Eclipse* com *dashboard*, barra de ferramenta na parte superior, com o layout de pastas no canto esquerdo e com o painel de programação no lado direito, em que está escrito o seguinte código:

```

package javajdbc; public class DriverManger {}
```



Em seguida, crie outra classe, agora chamada *DriverManager.java*.



The screenshot shows the Eclipse IDE interface. The title bar reads "eclipse-workspace - projetojava/src/javajdbc/Connection.java - Eclipse IDE". The menu bar includes File, Edit, Source, Refactor, Navigate, Search, Project, Run, Window, and Help. The toolbar has various icons for file operations. On the left, the Package Explorer shows a project named "projetojava" with a "src" folder containing a "javajdbc" package that includes "Bancoacesso.java", "Connection.java", and "DriverManager.java". Other files like "projetojava.java", "module-info.java", and library references are also listed. The main workspace on the right displays the Java code for "Connection.java":

```
1 package javajdbc;
2
3 public class Connection {
4
5 }
6
7
8 |
```

**#PraCegoVer:** na imagem, temos um *print* da tela do software Eclipse com *dashboard*, barra de ferramenta na parte superior, com o layout de pastas no canto esquerdo e com o painel de programação no lado direito, em que está escrito o seguinte código:

```
package javajdbc; public class Connection { }
```

# Referências

BARRETO, R. **Aprendendo Java**. Joinville: Clube de Autores, 2015.

BURD, B. **Começando a programar em Java para leigos**. Rio de Janeiro: Alta Books, 2014.

DEITEL, P. J.; DEITEL, H. **Java: how to program**. 11. ed. São Paulo: Pearson, 2017.

FREEPIK - VETORES, FOTOS DE ARQUIVO E DOWNLOADS PSD GRÁTIS. <https://br.freepik.com/home>. [S.I.]. Freepik, 2021. Disponível em: <https://br.freepik.com/home>. Acesso em: 9 set. 2021.

KBASE. <https://kbase.com.br/>. [S.I.]. KBASE, 2021. Disponível em: <https://kbase.com.br/>. Acesso em: 20 out. 2021.

ORACLE Database XE Downloads. **Oracle**, [s. l.], 20 fev. 2019. Disponível em: <https://www.oracle.com/br/database/technologies/xe-downloads.html>. Acesso em: 30 abr. 2021.

PEXELS - AS MELHORES FOTOS PROFISSIONAIS GRATUITAS E IMAGENS E VÍDEOS LIVRES DE ROYALTIES QUE OS CRIADORES COMPARTILHARAM.. <https://www.pexels.com/pt-br/>. [S.I.]. Pexels, 2021. Disponível em: <https://www.pexels.com/pt-br/>. Acesso em: 11 nov. 2021.

SAKURAI, R. G. Java: tipos primitivos. **Universidade Java**, [s. l.], 15 jun. 2011. Disponível em: <http://www.universidadejava.com.br/materiais/java-tipos-primitivos/>. Acesso em: 30 abr. 2021.

SCHILDT, H. **Java para iniciantes**. 6. ed. Porto Alegre: Bookman, 2015.

SIERRA, K.; BATES, B. **Use a cabeça! JAVA**. 2. ed. Rio de Janeiro: Alta Books, 2007.

WEBTECH - WEBSITE INFORMATION AND TECHNOLOGY STACK. <https://webtechsurvey.com/website/media-exp1.licdn.com>. [S.I.]. Webtech, 2021. Disponível em: <https://webtechsurvey.com/website/media-exp1.licdn.com>. Acesso em: 14 out. 2021.

# Fechamento

Parabéns! Você finalizou o curso de Java Avançado.

Com os conhecimentos adquiridos, agora você consegue criar tudo o que precisa com a linguagem de programação Java, tendo segurança em suas ações e a confiança de que os elementos necessários estarão em sua programação.

Contudo, destacamos que é preciso sempre estar atento e manter-se atualizado, pois a linguagem Java se modifica com as novas tecnologias, a fim de atender às demandas do mercado e dos usuários!

Sempre que precisar, você pode retornar e rever o curso.

Boa sorte em sua jornada.

