**01Nosso primeiro método**

**Transcrição**

Estamos prontos para evoluir o nosso modelo de classes, bem como os objetos que estamos construindo a partir delas.

Em nossa classe Conta nós temos quatro atributos.

public class Conta {

double saldo;

int agencia;

int numero;

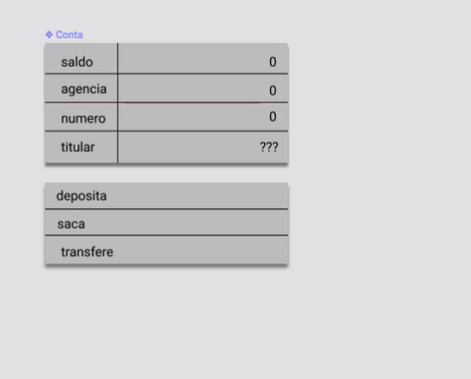
String titular;

}Copiar código

Sabemos que no momento em que acionamos a palavra-chave new, objetos são criados e os atributos referentes a esses objetos são zerados, inclusive os de tipo String.

Além de modificarmos os atributos de um determinado objeto, criaremos algumas funcionalidades. No caso de uma conta bancária é estranho que simplesmente surja 300 reais de saldo do nada. O interessante é que haja uma funcionalidade de *depósito* que inclua valores ao atributo saldo. Portanto, criaremos certos comportamentos para o objeto "conta bancária".

Podemos ordenar para a nossa conta funções como *sacar*, *depositar* e *transferir*.



Nossa conta, então, possuirá quatro atributos e três funções. Quando formos transferir essas informações para o Java, chamaremos essas funções de **métodos**, ou seja, uma maneira de fazer algo. Veremos posteriormente quais são as diferenças entre o termo "função" utilizado em outras linguagens e "método" no Java.

Escreveremos o método deposita() em nossa classe Conta. Atenção para a sintaxe utilizada: no parênteses () adicionaremos o que está sendo recebido pelo método, ou seja, um **parâmetro**. No caso de uma conta bancária, precisamos adicionar um valor a ser depositado. Em Java não se pode declarar uma variável sem especificar seu tipo, portanto, especificaremos a variável valor como sendo do tipo double.

public class Conta {

double saldo;

int agencia;

int numero;

String titular;

deposita(double valor)

}Copiar código

Depois que depositamos um valor em uma determinada conta bancária, poderemos receber uma mensagem, um número, uma espécie de comprovante ou algo do gênero. No caso do nosso projeto **ByteBank**, não há qualquer tipo de retorno à ação de depósito. Quando não existe qualquer tipo de retorno ao acionarmos um método, utilizamos a palavra-chave void. Feito isso, fecharemos o bloco utilizando as chaves {}

public class Conta {

double saldo;

int agencia;

int numero;

String titular;

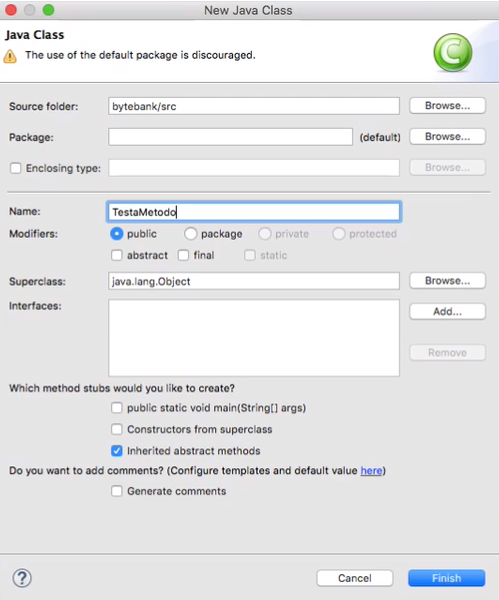
void deposita(double valor) {

}

}Copiar código

O método deposita() não produz nenhum resultado por enquanto, mas sua sintaxe é válida.

Agora que já conhecemos a sintaxe, aprenderemos como acionar e utilizar os métodos em Java. Criaremos uma nova classe intitulada TestaMetodo



Para invocarmos o método deposita(), é necessário nos referenciar à uma conta específica, neste caso, usaremos uma variável chamada de contaDoPaulo.

Lembrem-se: é comum o nome de uma variável ser igual ao da classe, sendo que a variável por convenção é escrita com letra minúscula.

public class TestaMetodo {

public static void main(String[] args) {

Conta contaDoPaulo = new Conta();

}

}Copiar código

O saldo de contaDoPaulo terá um valor de 100. Para invocarmos o método deposita() utilizaremos o caractere ponto . seguindo dos parênteses que contém o valor que queremos depositar, que no caso será 50.

public class TestaMetodo {

public static void main(String[] args) {

Conta contaDoPaulo = new Conta();

contaDoPaulo.saldo = 100;

contaDoPaulo.deposita(50);

}

}Copiar código

Nosso método é válido, e mesmo não gerando nenhum resultado, poderá ser executado sem nenhum erro. Podemos realizar um teste adicionando o Sysout a nossa classe TestaMetodo:

public class TestaMetodo {

public static void main(String[] args) {

Conta contaDoPaulo = new Conta();

contaDoPaulo.saldo = 100;

contaDoPaulo.deposita(50);

System.out.println(contaDoPaulo.saldo);

}

}Copiar código

Ao executarmos a aplicação veremos que o resultado impresso será 100, ou seja, não foi depositado nenhum novo valor à contaDoPaulo. Nada ocorreu, pois ao evocarmos o método deposita estamos nos referindo ao código escrito na classe Conta, e este código não executa nada, como já sabemos.

public class Conta {

double saldo;

int agencia;

int numero;

String titular;

void deposita(double valor) {

}

}Copiar código

Queremos que o método deposita() adicione valores à uma determinada conta. Existem muitas maneiras de escrevermos esse código, mas faremos da seguinte forma:

Adicionamos o public ao método deposita(), não se atente para isso neste momento do curso, discutiremos essa questão posteriormente

public class Conta {

double saldo;

int agencia;

int numero;

String titular;

public void deposita(double valor) {

saldo = saldo + valor;

}

}Copiar código

Reparem que há uma diferença de cores entre saldo e valor, pois o primeiro é um atributo do objeto Conta, enquanto o segundo é uma variável.

Feitas estas alterações, retornaremos à classe TestaMetodo e executaremos a aplicação.

public class TestaMetodo {

public static void main(String[] args) {

Conta contaDoPaulo = new Conta();

contaDoPaulo.saldo = 100;

contaDoPaulo.deposita(50);

System.out.println(contaDoPaulo.saldo);

}

}

Copiar código

Teremos, dessa vez, um valor impresso de 150. O método deposita() alterou o valor do atributo saldo, que é uma característica de conta.

Para deixarmos mais clara essa noção, usaremos outra palavra-chave do Java que é opcional neste caso específico, o this. Ao observarmos classe Conta, veremos que no bloco do método deposita() há uma referência de saldo, mas qual saldo? De que conta estamos falando?

public class Conta {

double saldo;

int agencia;

int numero;

String titular;

public void deposita(double valor) {

saldo = saldo + valor;

}

}Copiar código

Podemos escrever nosso código da seguinte forma:

public class Conta {

double saldo;

int agencia;

int numero;

String titular;

public void deposita(double valor) {

contaDoPaulo = saldo + valor;

}

}

Copiar código

Não conseguiremos compilar nosso código desta forma, pois contaDoPaulo não é uma variável presente neste escopo.Queremos que saldo seja relacionado à conta que está evocando o método deposita(), para isso, faremos uso da palavra-chave this.

public class Conta {

double saldo;

int agencia;

int numero;

String titular;

public void deposita(double valor) {

this.saldo = this.saldo + valor;

}

}Copiar código

Como o método está sendo invocado pela contaDoPaulo, o saldo é referente a esta conta. Não incluímos a palavra-chave this junto à variável valor, pois ela **não** é um atributo de um objeto.

Criamos, assim, nosso primeiro método. Escrevemos deposita(), parâmetros e qual será a devolução gerada pelo método.

**02Sobre métodos**

O que aprendemos sobre métodos?

Parte superior do formulário

* Um método define o comportamento ou a maneira de fazer algo.

Correto, esse é o objetivo de métodos, definir o que um objeto sabe fazer. O comportamento é implementado dentro do método.

* Alternativa correta

Um método **não** precisa definir a saída. Se não tiver um valor de retorno, basta escrever apenas o nome de método.

* Alternativa correta

É possível que um método não tenha nenhum parâmetro.

Correto, um método pode ter nenhum, um ou mais parâmetros.

* Alternativa correta

Por convenção, o nome do método no mundo Java deve começar com letra minúscula.

Correto, exemplo de nome dos métodos são:

transfere

transferePara

transfereParaOutroTitularCopiar código

Repare que todos os nomes começam com letra minúscula e depois usam "CamelCase".

Parte inferior do formulário

Parabéns, você acertou!

**03Como chamar um método?**

Qual é a sintaxe e ordem correta para chamar um método com Java?

Parte superior do formulário

* nomeDoMetodo.nomeDaReferencia();
* Alternativa correta

nomeDaReferencia -> nomeDoMetodo();

* Alternativa correta

nomeDaReferencia.nomeDoMetodo;

* Alternativa correta

nomeDaReferencia.nomeDoMetodo();

Correto, sempre usando nomeDaReferencia.nomeDoMetodo(); , por exemplo:

Conta contaDoPaulo = new Conta();

contaDoPaulo.deposita(30.0);Copiar código

Parte inferior do formulário

Parabéns, você acertou!

Este conteúdo foi útil para o seu aprendizado?

Sim

Não muito

**04Você conhece o this?**

Todas as afirmações abaixo são verdadeiras exceto:

Parte superior do formulário

* O this é uma referência.
* Alternativa correta

O uso do this é obrigatório.

Não é verdade, é justamente contrário. O uso do this é opcional, mas recomendamos o seu uso, principalmente para o iniciante na linguagem Java.

* Alternativa correta

O this é uma palavra chave.

Parte inferior do formulário

Parabéns, você acertou!

Este conteúdo foi útil para o seu aprendizado?

Sim

Não muito

**05Métodos com retorno**

**Transcrição**

Trabalharemos em um método mais elaborado. O deposita() que criamos, não devolvia nenhum tipo de informação para quem o invocou.

public class TestaMetodo {

public static void main (String[] args) {

Conta contaDoPaulo = new Conta();

contaDoPaulo.saldo = 100;

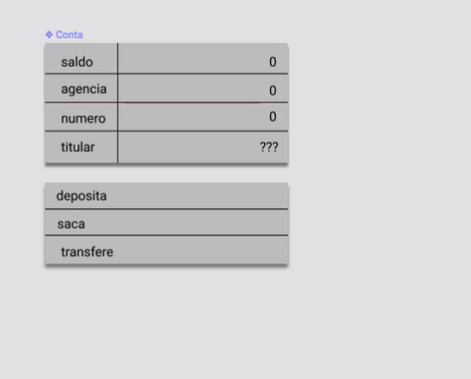
contaDoPaulo.deposita(50);

System.out.println(contaDoPaulo.saldo);

}

}Copiar código

Ao observarmos o diagrama, notamos que existe um método chamando saca(), que tem por função *retirar dinheiro de uma conta específica*.



Criaremos o novo método saca() na classe Conta. Para realizarmos um saque, é preciso atribuir um valor, - uma variável do tipo double- ao saque.

public class Conta {

double saldo;

int agencia;

int numero;

String titular;

public void deposita(double valor) {

this.saldo = this.saldo + valor;

}

saca(double valor);

}Copiar código

Não precisamos ter a informação de qual conta iremos sacar neste ponto, e essa é uma diferença da forma de programar orientada a métodos e a orientada a funções. Em métodos sempre há um "sujeito" do código à esquerda ( no exemplo, contaDoPaulo ), de forma que sabemos o direcionamento de determinado comando.

contaDoPaulo.saldo = 100 Copiar código

Iremos fazer com o que o método saca() nos retorne um boolean: true caso o saque seja efetivado, false caso não.

Para isso, escreveremos o public e boolean na linha do método e fechamos as chaves {}. Reparem que o método deposita() está em um bloco e método saca() em outro, não existe método dentro de método.

public class Conta {

double saldo;

int agencia;

int numero;

String titular;

public void deposita(double valor) {

this.saldo = this.saldo + valor;

}

public boolean saca(double valor) {

}

}Copiar código

Da forma como escrevemos o código ele não será compilado, pois acionamos um boolean sem escrever qual seria o retorno dado pelo método. Para prosseguirmos com o nosso código, acionaremos a palavra-chave if. Usaremos , também, o this, que aciona a referência para a conta que está acionando o método. Se o saldo de this for >= ao valor de saque, o novo saldo será o saldo de this - valor de saque. O método, então, retornará o valor true. Caso o contrário, ( else) será retornado o valor false.

public class Conta {

double saldo;

int agencia;

int numero;

String titular;

public void deposita(double valor) {

this.saldo = this.saldo + valor;

}

public boolean saca(double valor) {

if(this.saldo >= valor) {

this.saldo = this.saldo - valor;

return true;

} else {

return false;

}

}

}Copiar código

Iremos na classe TestaMetodo para analisarmos se o nosso código está funcional. Iremos sacar 20 reais de contaDoPaulo:

public class TestaMetodo {

public static void main (String[] args) {

Conta contaDoPaulo = new Conta();

contaDoPaulo.saldo = 100;

contaDoPaulo.deposita(50);

System.out.println(contaDoPaulo.saldo);

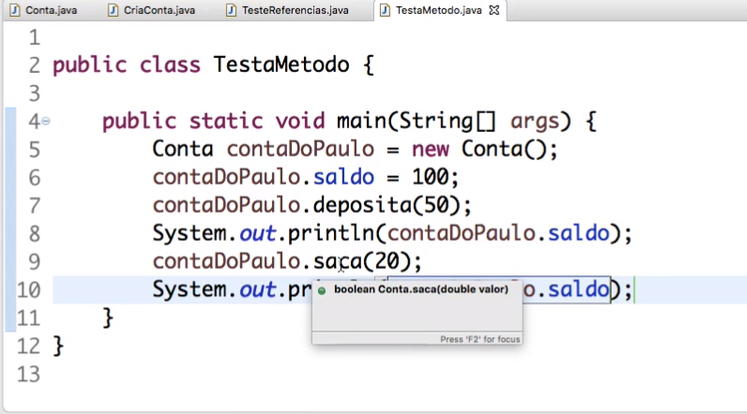
contaDoPaulo.saca(20);

System.out.println(contaDoPaulo.saldo);

}

}Copiar código

Ao rodarmos a aplicação, veremos que o resultado será 130. Ou seja, o nosso comando de saque foi efetivo. A respeito do true e false do boolean, quando passamos o mouse sobre o código vemos o retorno do método, podemos ou não utilizar esse valor.



Podemos guardar esse valor dentro de uma variável que chamaremos de conseguiuRetirar, e acionaremos o sysout.

public class TestaMetodo {

public static void main (String[] args) {

Conta contaDoPaulo = new Conta();

contaDoPaulo.saldo = 100;

contaDoPaulo.deposita(50);

System.out.println(contaDoPaulo.saldo);

boolean conseguiuRetirar = contaDoPaulo.saca(20);

System.out.println(contaDoPaulo.saldo);

System.out.println(conseguiuRetirar);

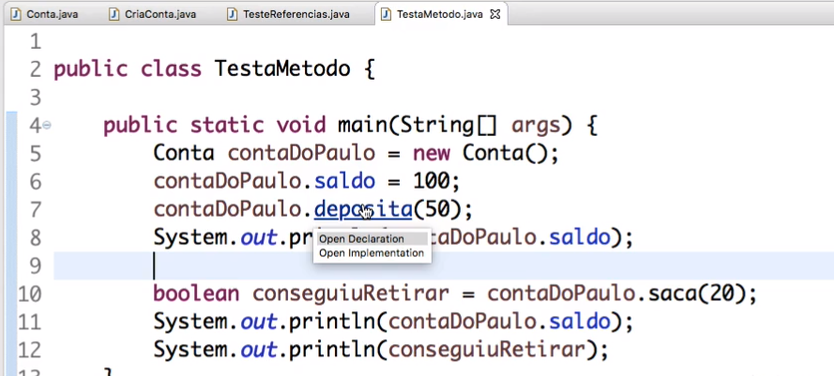
}

}Copiar código

Ao executarmos o programa, veremos que o resultado será true.

Abordaremos algumas melhorias de navegação que podemos executar no Eclipse. Normalmente, deixamos muitas abas abertas no editor, uma forma mais rápida e fácil de transitar no projeto que está sendo desenvolvido é manter pressionado atalho "Ctrl", com isso, o Eclipse irá transformar muitos elementos em links.

Ao clicarmos em um dos links, seremos transportados para o código referente ao método ou classe que estamos acionando.



Outra melhoria que podemos fazer quanto a sua formatação, é alterar a linha this.saldo = this.saldo + valor utilizando +=, e na linha this.saldo = this.saldo - valor fazer uso de -=.

Essas alterações deixarão o código mais enxuto, e além de ser mais comum na comunidade de programadores Java este tipo de formato, o resultado final será o mesmo.

public void deposita(double valor) {

this.saldo += valor;

}

public boolean saca(double valor) {

if(this.saldo >= valor) {

this.saldo -= valor;

return true;

} else {

return false;

}

} Copiar código

**06Métodos validos**

Assumindo que cada método abaixo está dentro de uma classe, quais declarações estão válidas (compilam)?

Parte superior do formulário

* void deposita(double valor, int numero)
* Alternativa correta
* deposita(double valor){

}

* Alternativa correta
* void deposita(double valor, int numero){

}

Correto, definimos o retorno (nesse caso void), definimos o nome do método (nesse caso deposita) e definimos corretamente os parâmetros (nesse casos dois, double valor e int numero ) .

* Alternativa correta
* void deposita(){

}

Correto, definimos o retorno (nesse caso void), definimos o nome do método (nesse caso deposita) e definimos corretamente os parâmetros (nesse casos não temos nenhum parâmetro ) .

Parte inferior do formulário

Parabéns, você acertou!

**07Onde usar o this?**

A Ana está praticando OO com Java e criou uma outra classe Conta com apenas dois atributos e um método. No entanto, como o uso do this é opcional, ela está com dúvida onde se usa a palavra chave this dentro de uma classe.

Ela nos enviou o código abaixo usando os caracteres [] em vários lugares, segue o código:

class Conta {

[1] double saldo;

int numero;

void deposita([2] double valor) {

[3]saldo = [4]saldo + [5]valor;

}

}Copiar código

Repare que temos [1], [2], [3], [4] e [5] como possíveis lugares para colocar o this, mas qual realmente irá funcionar e ser compilado?

Parte superior do formulário

* Apenas [1] e [2]
* Alternativa correta

[1], [2] e [3]

* Alternativa correta

Apenas [5]

* Alternativa correta

Apenas [1]

* Alternativa correta

Apenas [3] e [4]

Usamos o this dentro de um método para acessar um atributo:

class Conta {

double saldo;

int numero;

void deposita(double valor) {

this.saldo = this.saldo + valor;

}

}Copiar código

Parte inferior do formulário

Parabéns, você acertou!

Este conteúdo foi útil para o seu aprendizado?

Sim

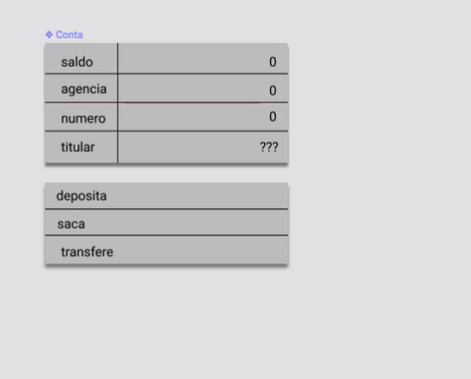
Não muito

**08Métodos com referência e mais retorno**

**Transcrição**

Tenho um desafio para que você saiba se entendeu o assunto **métodos**.

De acordo com o diagrama de contas, chegamos ao nosso terceiro método: o transfere(), que tem como finalidade **transferir dinheiro** de uma conta para outra.



Vamos incluir o novo método na classe Conta.

Abaixo do método saca(), adicionaremos transfere().

Pensaremos sobre um ponto: quando transferimos dinheiro de uma conta para outra, receberemos uma informação de volta? Neste caso, sim. Para que uma transferência seja realizada com sucesso, é necessário haver dinheiro o suficiente em uma conta. Caso contrário, o retorno será false.

Usaremos o tipo boolean, com uma finalidade parecida com a do método saca().

public boolean saca(double valor){

if(this.saldo >= valor) {

this.saldo -= valor;

return true;

} else {

return false;

}

}

public boolean transfere() {

}Copiar código

E quais argumentos o método recém-criado receberá? Nós, certamente, transferiremos um valor, expressaremos isso adicionando a variável valor que será do tipo double.

public boolean transfere(double valor, Conta destino) {

}Copiar código

Observe que adicionamos um segundo argumento, que foi separado por uma vírgula (,) do primeiro. Foi necessário incluir um segundo argumento no método referente a **conta de destino** do depósito, ou seja, a conta para onde o dinheiro será transferido.

No entanto, será que é preciso especificar a conta de onde o dinheiro será retirado? Quantos argumentos serão necessários?

Nós adicionamos a Conta destino. Pela primeira vez, recebemos uma variável que usa também letras maiúsculas. Falta nela a tipagem de double ou int como fizemos anteriormente. Agora, estamos fazendo uma referência para outra conta.

Será que devemos receber a Conta origem? Não. E por quê?

Antes, você vai perceber que a linha com transfere() está vermelha. Isto ocorre, porque nosso código não está sendo compilado, afinal, falta adicionar o retorno (return).

Em seguida, acessaremos TestaMetodo.java. Abaixo do último println de conseguiuRetirar, criaremos uma referência a uma outra conta, que chamaremos de contaDaMarcela.

Será nesta conta que depositaremos 1000 reais. Nosso objetivo será transferir 300 reais de contaDaMarcela para contaDoPaulo.

O interessante da orientação a objetos é que a nossa primeira frase será uma *referência* para a variável.

public class TestaMetodo {

public static void main(String[] args) {

Conta contaDoPaulo = new Conta();

contaDoPaulo.saldo = 100;

contaDoPaulo.deposita(50);

System.out.println(contaDoPaulo.saldo);

boolean conseguiuRetirar = contaDoPaulo.saca(20);

System.out.println(contaDoPaulo.saldo);

System.out.println(conseguiuRetirar);

Conta contaDaMarcela = new Conta();

contaDaMarcela.deposita(1000);

}

}Copiar código

Primeiramente, escreveremoscontaDaMarcela e depois, passaremos o método transfere().

Enquanto digitamos o código, o *autocomplete* do Eclipse vai disponibilizar diversos métodos, incluindo transfere(), que ainda tem um erro no outro arquivo.

No entanto, se tentássemos já executar o código, receberíamos um aviso de possível problema na compilação.

Como parâmetro, passaremos o valor de 300, que será transferido, e como parâmetro de destino, incluiremos uma referência de contaDoPaulo. Foi desnecessário colocar, por exemplo, o id da conta do Paulo. O método vai receber um variável do tipo conta, que não é um objeto Conta.

Você pode ter ficado com a impressão de que estamos passando uma conta dentro de um método. Este não é o caso. Nós mandamos um "número interno" que o Java enxerga e não precisa tanto da nossa atenção.

public class TestaMetodo {

public static void main(String[] args) {

Conta contaDoPaulo = new Conta();

contaDoPaulo.saldo = 100;

contaDoPaulo.deposita(50);

System.out.println(contaDoPaulo.saldo);

boolean conseguiuRetirar = contaDoPaulo.saca(20);

System.out.println(contaDoPaulo.saldo);

System.out.println(conseguiuRetirar);

Conta contaDaMarcela = new Conta();

contaDaMarcela.deposita(1000);

contaDaMarcela.transfere(300, contaDoPaulo);

}

}Copiar código

Evitamos adicionar ContaDaMarcela, porque a conta de origem é o valor no primeiro parâmetro. Ou seja, vamos passar apenas a conta de destino, considerando que a origem já é o objeto ao qual invocamos o método.

Seria equivalente a referência this do método transfere() no outro lado. Agora, só falta implementá-lo na classe Conta.

Para isto, usaremos if: caso o saldo de this - fazendo referência a contaDaMarcela - seja igual ou superior ao valor de transferência, será subtraído de saldo o valor referente à transferência.

public class Conta {

// atributos

// métodos

public boolean transfere(double valor, Conta destino) {

if(this.saldo >= valor) {

this.saldo -= valor;

}

}

}Copiar código

Por enquanto, apenas retiramos um valor de saldo, precisaremos que este valor seja transferido para uma conta destino, no caso, contaDoPaulo. Existem duas formas de executar essa função, uma delas é esta:

destino.saldo += valor; Copiar código

Ou, podemos reutilizar um método da classe Conta, o deposita().

Retiramos um valor e ele foi depositado. Neste caso, a transferência foi realizada com sucesso, e o retorno será true. Caso não haja dinheiro o suficiente para realizar a transferência, será retornado false.

A utilização do else é opcional no código, o retorno false ocorrerá mesmo que esta palavra não tenha sido utilizada.

public class Conta {

// atributos

// métodos

public boolean transfere(double valor, Conta destino) {

if(this.saldo >= valor) {

this.saldo -= valor;

destino.deposita(valor);

return true;

} else {

return false;

}

}

}Copiar código

Em TestaMetodo, acionaremos o sysout para contaDaMarcela.

public class TestaMetodo {

public static void main(String[] args) {

Conta contaDoPaulo = new Conta();

contaDoPaulo.saldo = 100;

contaDoPaulo.deposita(50);

System.out.println(contaDoPaulo.saldo);

boolean conseguiuRetirar = contaDoPaulo.saca(20);

System.out.println(contaDoPaulo.saldo);

System.out.println(conseguiuRetirar);

Conta contaDaMarcela = new Conta();

contaDaMarcela.deposita(1000);

contaDaMarcela.transfere(300, contaDoPaulo);

System.out.println(contaDaMarcela.saldo);

}

}Copiar código

Ao executarmos o programa veremos que o valor impresso será 700, ou seja, nosso código funcionou perfeitamente.

Podemos, inclusive, verificar se no saldo de contaDoPaulo há 300 reais a mais, como o esperado. Para isso. acionaremos o sysout novamente para contaDoPaulo.

contaDaMarcela.transfere(300, contaDoPaulo);

System.out.println(contaDaMarcela.saldo);

System.out.println(contaDoPaulo.saldo); Copiar código

Veremos que o valor impresso ao final da execução será de 430 reais, comprovando o sucesso do método. Não utilizamos o valor devolvido pelo *boolean*, poderíamos ter guardado esse valor através da utilização de if. Estamos invocando o método transfere() - lembre-se que dentro de if só são possíveis expressões booleanas - que devolve *boolean*, e portanto, o if pode ser compilado.

Caso o resultado dessa transferência tenha dado true, iremos imprimir "transferência com sucesso". Caso contrário, imprimiremos "faltou dinheiro".

public class TestaMetodo {

public static void main(String[] args) {

Conta contaDoPaulo = new Conta();

contaDoPaulo.saldo = 100;

contaDoPaulo.deposita(50);

System.out.println(contaDoPaulo.saldo);

boolean conseguiuRetirar = contaDoPaulo.saca(20);

System.out.println(contaDoPaulo.saldo);

System.out.println(conseguiuRetirar);

Conta contaDaMarcela = new Conta();

contaDaMarcela.deposita(1000);

if(contaDaMarcela.transfere(300, contaDoPaulo)) {

System.out.println("transferencia com sucesso");

} else {

System.out.println("faltou dinheiro");

}

System.out.println(contaDaMarcela.saldo);

System.out.println(contaDoPaulo);

}

}Copiar código

Caso a linha que inserimos o if fique muito grande ou complicada, é normal quebrarmos em duas linhas. Poderíamos alocar a parte do código referente ao método transfere() e inserir uma variável sucessoTransferencia em seu lugar.

public class TestaMetodo {

public static void main(String[] args) {

Conta contaDoPaulo = new Conta();

contaDoPaulo.saldo = 100;

contaDoPaulo.deposita(50);

System.out.println(contaDoPaulo.saldo);

boolean conseguiuRetirar = contaDoPaulo.saca(20);

System.out.println(contaDoPaulo.saldo);

System.out.println(conseguiuRetirar);

Conta contaDaMarcela = new Conta();

contaDaMarcela.deposita(1000);

boolean sucessoTransferencia = contaDaMarcela.transfere(300, contaDoPaulo);

if(sucessoTransferencia) {

System.out.println("transferencia com sucesso");

} else {

System.out.println("faltou dinheiro");

}

System.out.println(contaDaMarcela.saldo);

System.out.println(contaDoPaulo.saldo);

}

}Copiar código