**Java OO: Introdução à orientação a Objetos**

01 – O problema do paradigma procedural

- o ideal é que possamos fazer uma alteração em um único local do sistema e essa alteração reflita em todos os lugares que seja necessário

- diferente do paradigma procedural, a orientação a objetos, dados e funcionalidades (comportamentos) estarão interligados, gerando uma grande facilidade na organização e manutenção

- ideia central da orientação a objeto

- dado e funcionalidade andam juntos, exemplo:

- se um dados qualquer (CPF) por exemplo estivesse atrelado a uma função de validação, poderíamos dizer que aplicamos a OO

- para que varia equipes consigam trabalhar em um mesmo projeto é necessário que responsabilidades de cada parte do codigo estejam bem definidas e claras, evitando conflitos e erros na horas de implementarmos mudanças no mesmo

- códigos com responsabilidades coesas é um sinal do paradigma OO

- alguns exemplos da utilização do paradigma procedural

- regras de negocio espalhadas por diversos locais

- utilização de codigo pronto (tipo copiar e colar) para assim evitar erros dos códigos que estão funcionando em produção

- com a regra espalhada em diversos locais cresce a complexidade de manutenção

- diversas pessoas mexendo em funções parecidas porem de setores distintos acabavam criando códigos redundantes, pois fazem basicamente a mesma função, logo temos trechos de códigos espalhados por todo o sistema realizando a mesma coisa e com isso dificultando manutenção.

02 – Introduçao á Orientaçao a Objetos

- primeira classe – contas

- além dos dados da conta também armazenaremos comportamentos da conta (saques, depósitos, transferências, etc)

- a palavra ***class***

- é uma palavra reservada do Java que define um tipo

- para indicar que abrimos e fechamos um bloco de informações utilizamos { }

- no Java é necessário declarar o tipo (***double, int, String***) por exemplo

- CTRL + S salva o trabalho e com isso gera um codigo fonte valido em Java que já foi compilado

- dentro do diretório bin teremos o arquivo .class

- para que um arquivo possa ser iniciado no Java é necessário um ponto de entrada ***public static void main (String [] args)***

- classes que não possuem o atributo acima são arquivos de suporte

- o primeiro arquivo a ser executado no Java contem o método ***main*** com o tipo de retorno e parâmetros

- instanciação, atributos e referencias

- no exemplo temos uma conta bancaria com 4 características (saldo, agencia, numero e titular), essas caracteristicas chamamos de atributos

***class Conta {***

***double saldo;***

***int agencia;***

***int numero;***

***String titular;***

***}***

- o que temos acima ainda não é uma conta bancaria e sim uma especificação de uma conta, pois ainda não podemos realizar as operações básicas que envolvem uma conta - a partir dessa especificação podemos construir varias contas bancarias individuais

- a partir da especificação da conta (classe) construiremos objetos ou instancias do tipo conta

- a instanciação é transformar uma classe em um objeto (no exemplo objeto do tipo conta)

- fazendo um paralelo é semelhante a tirar uma casa da planta e construí-la

- dada classe conta instanciamos um objeto tipo conta

- podemos alterar os atributos do objeto (instancia) conta

- não podemos alterar as especificações do que é uma conta, mas sim as contas de uma forma individual

- criaremos no curso uma classe publica

***public class CriaConta {***

***public static void main (String [] args {***

***}***

***}***

- palavra chave  ***new***  e ao lado o nome da classe que servira para criação de objetos, no nosso caso a classe ***Conta()***

- temos um programa que dentro da nossa ***main***cria (instancia) ou constrói um objeto tipo ***Conta***

- guardamos o retorno de ***new Conta ()*** em uma variável ***primeiraConta***

- no Java o objeto nunca esta dentro de uma variável

- o que temos dentro de uma variável é somente uma indicação a um objeto especifico, uma referencia

***public class CriaConta {***

***public static void main (String [] args {***

***Conta primeiraConta = new Conta( );***

***primeiraConta.saldo = 200;***

***System.out.println(primeiraConta.saldo);***

***}***

***}***

P: Como é chamado em OO as caracteristicas de uma classe?

R: Atributo

- uma classe é uma especificação de um tipo, definindo atributos e comportamentos

- um objeto é uma instancia de uma classe onde podemos definir valores para seus atributos

- para criar uma instancia precisamos usar a palavra chave ***new***

- ao criarmos uma instancia não é obrigatório preencher os valores dos atributos

- segunda instancia

- além de atribuir valores utilizando o sinal de =, podemos utilizar += (significa que o resultado anterior + algo)

- no nosso exemplo a variável ***primeiraConta*** é referencia a uma conta especifica (objeto)

- criamos a variável ***segundaConta*** que fara referencia a outra conta

- instaciaremos através do nosso codigo base embutido em ***Conta*** que já possui alguns atributos (saldo, agencia, numero e titular)

- utilizaremos a palavra chave ***new*** e a informação devolvida sera armazenada dentro da referencia ***segundaConta***, que sera uma variável do tipo ***Conta***

- devemos sempre ***referencia.atributo***

***segundaConta.saldo***

***primeiraConta.agencia***

- essa é a forma de se trabalhar orientado a objeto

- valores default de atributos

- as variáveis que utilizamos no primeiro curso ficavam diretamente no método ***main*** e nao possuíam um valor default (padrão), não sendo possível sequer executar uma aplicação com esse tipo de variável pois não havia valores definidos

- as variáveis que estamos utilizando na classe ***Conta***, não são as mesmas que ficam dentro do método ***main***

- essas variáveis ficam armazenadas diretamente na classe

- chamamos essas variáveis de atributos ou caracteristicas de um objeto

- são similares como se comparássemos a uma planta de uma casa (numero de quartos, sala, numero de banheiros, etc)

- no caso de um conta bancaria seriam (agencia, saldo, conta, titularidade)

- esses atributos podem ser chamados de campo ou propriedade

- esse tipo de variável que possui significado para objetos possui comportamento diferente.

- quando utilizamos a palavra-chave ***new*** o Java instancia o objeto e todos o campos estão zerados

- quando o Java constrói um objeto todos os seus atributos são zerados

- 0 é o valor default de vários tipos numéricos (long, int, double por exemplo)

- no caso de uma variável tipo boolean o default é false

- podemos configurar valores diferentes de zero na classe ***Conta*** e posteriormente alterar esse valor no método ***main***

- um exemplo:

***public class Pessoa {***

***String nome;***

***int peso;***

***int idade;***

***}***

- uma opção para criar um objeto e definir um valor para seus atributos seria:

***Pessoa herói = new Pessoa();***

***herói.nome = “Rodrigo”;***

- no exemplo acima o objeto é criado, sua referencia é atribuída a variável herói e em seguida atribuímos um nome

- referencias VS objetos

- tipo ***Conta*** e a variável desse tipo ***primeiraConta***

- a variável não é um objeto ***Conta*** e sim uma indicação a um objeto especifico, uma referencia de um objeto

Interface gráfica do usuário, Texto, Aplicativo

Descrição gerada automaticamente

- no exemplo acima podemos pensar que a declaração ***segundaConta = primeiraConta*** gere uma copia do objeto ***primeiraConta***

- no Java o sinal = copia o que esta a direita e cola na esquerda

- quando realizamos a declaração ***segundaConta = primeiraConta***

- estamos realizando a copia do ID do objeto que é a referencia e não o objeto em si

- o que temos são duas referencias para o mesmo objeto

- como se duas cartas fossem endereçadas ao mesmo local, embora cartas diferentes possuem o mesmo destino

- o sinal == ira comparar referencias e não objetos

- Exercicio:

***public class Conta {***

***double saldo;***

***]***

***\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_***

***public class Teste {***

***public static void main (String [ ] args {***

***Conta minhaConta = new Conta ();***

***minhaConta.saldo = 500.0;***

***Conta outraConta = minhaConta***

***outraConta.saldo = 1000.0;***

***System.out.println(minhaConta.saldo);***

***}***

- o exemplo acima imprimira 1500.0 pois as duas referencias (***minhaConta*** e ***outraConta***) apontam para o mesmo objeto

**package** bytebank;

**public** **class** criaConta {

**public** **static** **void** main(String[] args) {

Conta primeiraConta = **new** Conta();

primeiraConta.saldo = 200.0;

System.***out***.println(primeiraConta.saldo);

primeiraConta.saldo += 100;

System.***out***.println(primeiraConta.saldo);

Conta segundaConta = primeiraConta;

segundaConta.saldo = 50;

Conta terceiraConta = **new** Conta();

terceiraConta.saldo = 1000;

System.***out***.println("A primeira conta tem de saldo R$" + primeiraConta.saldo);

System.***out***.println("A segunda conta tem de saldo R$" + segundaConta.saldo);

System.***out***.println("A terceira conta tem de saldo R$" + terceiraConta.saldo);

System.***out***.println(primeiraConta);

System.***out***.println(segundaConta);

System.***out***.println(terceiraConta);

}

}

Interface gráfica do usuário, Texto, Aplicativo

Descrição gerada automaticamente

03 – Definindo comportamento

- nosso primeiro método

- quando acionamos a palavra chave ***new***, objetos são criados e os atributos referentes a este objeto são zerados (inclusive os de tipo ***String***)

- criaremos comportamentos para o objeto do exemplo (conta bancaria)

- exemplos de comportamentos (funções) para conta (sacar, depositar, transferir, etc)

- nossa conta ate o momento possui 4 atributos:

- saldo

- agencia

- numero

- titular

- e possui 3 comportamentos (funções):

- saque

- deposito

- transferência

- as funções (métodos) no Java são maneira de realizar algo

- o método deposita sera criado na classe Conta

- deve ser colocado entre parênteses o parâmetro (o que esta sendo recebido pelo método deposita) e também temos que especificar o tipo da variável

***deposita (double valor)***

- após realizarmos o método no caso o deposita poderíamos receber algum retorno (mensagem, comprovante de deposito, etc) no nosso exemplo não receberemos nada de retorno

- quando não temos um retorno de um método utilizamos a palavra chave ***void***

- feito isso abrimos e fechamos o bloco com {}

**void** deposita(**double** valor) {

}

- para invocar o método deposita é necessário nesse caso referenciar a uma conta especifica

- é comum o nome da variável ser igual a da classe, porem a variável por convenção é escrita em minúscula

**public** **class** testaMetodo {

**public** **static** **void** main(String[] args) {

Conta contaDoRodrigo = **new** Conta();

contaDoRodrigo.saldo = 1000;

contaDoRodrigo.deposita(500);

}

}

- para invocarmos o método ***deposita()***

- utilizamos o caractere ponto . seguido dos parênteses que contem dentro o valor que queremos adicionar

- o codigo acima quando impresso no terminal retorna o valor de 1000, ou seja o valor de 500 não foi adicionado pois o método deposita da classe Conta é do tipo void (não executa nada

**public** **void** deposita(**double** valor) {

**this**.saldo = **this**.saldo + valor;

}

- no trecho acima o saldo é um atributo da conta enquanto o valor é uma variável

- a palavra reservada this faz referencia (relaciona) a conta que esta invocando o método

**public** **class** testaMetodo {

**public** **static** **void** main(String[] args) {

Conta contaDoRodrigo = **new** Conta();

contaDoRodrigo.saldo = 1000;

contaDoRodrigo.deposita(500);

System.***out***.println("O saldo da conta do Rodrigo é R$" + contaDoRodrigo.saldo);

Conta contaDoLucas = **new** Conta();

contaDoLucas.saldo = 10000;

contaDoLucas.deposita (800);

System.***out***.println("O saldo da conta do Lucas é R$" + contaDoLucas.saldo);

}

}

- recapitulando sobre métodos

- define um comportamento ou maneira de realizar algo

- definir o que um objeto sabe fazer esse comportamento é implementado dentro do metodo

- no mundo Java qualquer método precisa definir um retorno, mesmo quando não queremos retornar nada (nesse caso utilizamos a palavra void)

- pode não ter nenhum parâmetro

- como também pode ter mais de um

- por convenção o nome começa com letra minúscula

***deposita***

***transfereParaOutraConta***

***saca***

- quando composto de mais de uma palavra utilizamos o camelCase

- a sintaxe correta para chamar um método é:

***nomeDaReferencia.nomeDoMetodo();***

Conta contaDoLucas = **new** Conta();

contaDoLucas.deposita (800)

- em relação ao ***this***

- é uma referencia (aponta para algo)

- o seu uso não é obrigatório (mas recomendado para iniciantes)

- é uma palavra chave assim como: ***void, class, new, int***

- o eclispe mostra na cor roxa

- métodos com retorno

- criaremos o método saca na classe Conta

- atribuiremos um valor e um tipo (double)

**public** **boolean** saca(**double** valor) {

**if**(**this**.saldo >= valor) {

**this**.saldo -= valor;

**return** **true**;

} **else** {

**return** **false**;

}

- não precisamos nesse ponto ter a conta que sera sacado o dinheiro (uma diferença de OO para Orientado a função)

- em método sempre há um sujeito (a esquerda) 🡪 contaDoRodrigo 🡪 dessa forma o Java já sabe de quem vai realizar a operação (direcionamento)

***contaDoRodrigo.saldo = 100;***

***contaDoRodrigo.deposita(200);***

***contaDoRodrigo.saca(100);***

- o método saca retorna um boolean

- true caso seja efetivado o saque

- false caso não seja efetivado o saque

- o método deposita esta em um bloco de codigo e o método saca em outro bloco

- não há método dentro de método

- utilizamos o ***if*** para determinar se o ***this.saldo*** for maior ou igual valor o saque sera efetivado (true)

- ***else*** caso contrario não sera efetivado (false)

- retornando assim uma condição boleana

- utilizamos ao invés de:

***this.saldo = this.saldo – valor;***

***this.saldo -= valor;***

***this.saldo = this.saldo + valor;***

***this.saldo += valor;***

- isso deixa o codigo mais enxuto

- utilizamos o this dentro de um método para acessar um atributo

**public** **boolean** saca(**double** valor) {

**if**(**this**.saldo >= valor) {

**this**.saldo -= valor;

**return** **true**;

} **else** {

**return** **false**;

}

- no caso acima foi utilizamo no método saca para acessar o atributo saldo e realiza a operação de subtração.

\*\* o eclipse possui atalhos de navegação

- mantendo pressionada a tecla CTRL o eclipse transformara vários elementos (métodos, classe, etc) em links e quando clicamos em cima de algum deles somos direcionado para a aba onde esse elemento esta

Interface gráfica do usuário, Texto, Aplicativo

Descrição gerada automaticamente

- métodos com referencia e mais retorno

- adicionaremos o método transfere (já temos o saca e deposita)

- na operação de transferência recebemos uma informação de retorno nesse caso é necessário haver fundos na conta origem e o retorno sera true, caso contrario sera false

- esse método recebe dois argumentos

- valor (double)

- Conta destino

- separamos os argumentos por virgula

- esta variável ***Conta*** utiliza letra maiscula

- não temos a tipagem (double, int)

- essa variável na verdade faz referencia para onde o dinheiro sera transferido

- na OO o interessante que nossa primeira frase é uma referencia para uma variável

- o método vai receber uma variável do tipo conta, que não é um objeto Conta

- essa variável faz referencia a uma conta que foi instanciada (criada)

Conta contaDaPriscilla = **new** Conta();

contaDaPriscilla.deposita(5000);

contaDaPriscilla.transfere(300, contaDoLucas);

- evitamos colocar ContaDaPriscilla porque é a conta origem que é o valor do primeiro parâmetro, passamos apenas o destino, considerando que a origem já é o objeto ao qual invocamos o método

- nesse caso seria equivalente a referencia ***this***

**public** **boolean** transfere(**double** valor, Conta destino) {

**if**(**this**.saldo >= valor) {

**this**.saldo -= valor;

destino.saldo += valor;

}

- utilizamos o if caso o saldo da conta origem seja >= o valor sera realizado um -= valor (subtração do valor desejado) na conta origem e colocamos esse valor na conta destino +=valor

- caso haja dinheiro a operação é realizada com retorno true, caso contrario não sera realizada com retorno false

- a utilização do else é opcional no codigo pois o retorno false caso a condição não seja atendida no if ocorrera mesmo sem a utilização do else

**public** **class** testaMetodo {

**public** **static** **void** main(String[] args) {

Conta contaDoRodrigo = **new** Conta();

contaDoRodrigo.saldo = 1000;

contaDoRodrigo.deposita(500);

System.***out***.println("O saldo da conta do Rodrigo é R$" + contaDoRodrigo.saldo);

**boolean** conseguiuRetirar = contaDoRodrigo.saca(20);

System.***out***.println("O novo saldo da conta do Rodrigo é R$" + contaDoRodrigo.saldo);

System.***out***.println("O saque foi realizado com sucesso? " + conseguiuRetirar );

Conta contaDoLucas = **new** Conta();

contaDoLucas.saldo = 10000;

contaDoLucas.deposita (800);

System.***out***.println("\n" + "O saldo da conta do Lucas é R$" + contaDoLucas.saldo);

Conta contaDaPriscilla = **new** Conta();

contaDaPriscilla.deposita(5000);

**if** (contaDaPriscilla.transfere(500, contaDoLucas)) {

System.***out***.println("Transferencia realizada com sucesso!");

} **else** {

System.***out***.println("Transferencia não realizada, saldo insuficiente!");

}

System.***out***.println("\n" + "O saldo atual da conta da Prisilla é R$" + contaDaPriscilla.saldo);

System.***out***.println("O saldo da conta do Lucas é R$" + contaDoLucas.saldo);

}

}

**public** **class** Conta {

**double** saldo;

**int** agencia;

**int** numero;

String titular;

**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**;

}

}

**public** **boolean** transfere(**double** valor, Conta destino) {

**if**(**this**.saldo >= valor) {

**this**.saldo -= valor;

destino.saldo += valor;

**return** **true**;

}

**return** **false**;

}

}

Interface gráfica do usuário, Texto, Aplicativo

Descrição gerada automaticamente

Interface gráfica do usuário, Aplicativo

Descrição gerada automaticamente

- quando o método não retorna nenhum valor utilizamos void

04 – Composiçao de objetos

- o Java zera o valor dos atributos quando acionamos a palavra ***new***

- criamos um novo projeto ***bytebank-composto***

- uma classe ***Cliente***

- nome

- cpf

- profissão

- iremos estabelecer uma relação entre Conta e Cliente

- toda Conta faz uma referencia para um Cliente

- no novo projeto o atributo titular da classe Conta não sera mais uma String e sim uma referencia para um cliente especifico

- criamos uma nova classe ***TestaBanco***

- criamos um método ***main***

- instanciaremos o objeto Cliente e popularemos este objeto criando seus atributos (CPF, Nome, profissão)

**public** **class** TestaBanco {

**public** **static** **void** main(String[] args) {

Cliente rodrigo = **new** Cliente();

rodrigo.cpf = "215.548.698-70";

rodrigo.nome = "Rodrigo Sundfeld Batista de Moraes";

rodrigo.profissao = "Estudante";

Conta contaDoRodrigo = **new** Conta();

contaDoRodrigo.deposita(1000);

}

}

- a referencia para o cliente acima esta populada com os dados estipulados

- criamos também a conta do cliente e invocamos o método ***deposita***

- temos agora duas classes ***Conta*** e ***Cliente***

- o atributo ***titular*** passa de ***String***  para uma referencia para um objeto do tipo ***Cliente***

- na classe ***Conta*** alteramos o atributo ***titular*** para ser do tipo ***Cliente***

**public** **class** Conta {

**double** saldo;

**int** agencia;

**int** numero;

Cliente titular;

- agora temos o atributo ***titular*** fazendo referencia a um cliente especifico 🡪 associação entre objetos

- faremos essa associação na classe ***TestaBanco***

contaDoRodrigo.titular = rodrigo;

System.***out***.println(contaDoRodrigo.titular.nome);

System.***out***.println(contaDoRodrigo.titular);

- no primento print temos a seguinte saída

Rodrigo Sundfeld Batista de Morae

- no segundo print temos:

Cliente@5acf9800

- uma espécie de ID que possui o mesmo valor da variável ***rodrigo***

- Referencia Null

- nesta aula criamos uma classe ***TestaContaSemCliente***

- criamos todo o método ***main***

- criamos uma referencia para ***Conta*** chamada ***contaDoLucas*** utilizando a palavra chave ***new***

**public** **class** TestaContaSemCliente {

**public** **static** **void** main(String[] args) {

Conta contaDoLucas = **new** Conta();

System.***out***.println(contaDoLucas.saldo);

}

}

- o codigo acima é compilado e o retorno para saldo é 0.0

- porem se tentarmos realizar um procedimento parecido com ***titular*** e sem definir um ***Cliente*** teremos uma exception no retorno

Interface gráfica do usuário, Texto, Aplicativo

Descrição gerada automaticamente

- o “Zero” de um atributo ou variável do tipo referencia chamamos de null

- no codigo podemos ter uma referencia para nada (null) no codigo

System.***out***.println(contaDoLucas.titular);

- no print acima teremos como retorno a palavra ***null***

- o nome ***(titular)*** não faz referencia a nenhum ***Cliente***

- precisamos criar um novo cliente e fazer a associação ***Conta 🡪 Cliente***

- no codigo já temos essa associação através da variável ***contaDoLucas***

- temos que associar ***titular 🡪 nome***

- faremos que ***titular*** deixe de ser null fazendo-o receber um novo cliente: ***new Cliente***

contaDoLucas.titular = **new** Cliente();

- há casos que não existe necessidade de criar uma variável temporária podemos associar em uma linha

- normalmente no Java temos uma rede de objetos interconectados que se referenciam, e através da invocação de métodos conseguimos fazer com que trabalhem entre si

- temos como resultado códigos curtos que atuam em grandes conjuntos

- temos uma melhor organização do projeto (códigos fáceis de ler e de realizar manutenções)

- uma referencia é tida como null pois não foi populada ainda

- para popular uma referencia basta inserir um valor dentro dela

- através de ***new***

- ou apontando para uma referencia já existente

- podemos ter valores default exemplo, toda vez que uma conta é aberta ela inicia com valor de 100

- quando tentamos acessar uma propriedade de um objeto que não foi inicializado ele é null e resulta em um erro ***NullPointerException*** pois o ponteiro (referencia) esta apontado para lugar nenhum

05 – Encapsulamento e visibilidade

- atributos privados e encapsulamento

- criamos uma nova classe ***TestaSaldoNegativo***

- o tipo ***Conta*** deve ser o mesmo nome de uma classe que ela faz referencia

- a variável pode ter qualquer nome e por convenção escrevemos em minúsculo

- é comum que a variável tenha o mesmo da classe

**public** **class** TestaSaldoNegativo {

**public** **static** **void** main(String[] args) {

Conta contaDaPriscilla = **new** Conta();

contaDaPriscilla.deposita(100);

System.***out***.println(contaDaPriscilla.saldo);

contaDaPriscilla.saca(200);

System.***out***.println(contaDaPriscilla.saldo);

}

}

- quando tentamos sacar 200 não sera possível pois o saldo é 100 e a operação (método) saque só é true caso saldo >= valor

- podemos passar o valor booleano diretamente para o sysout, não precisamos sempre guardar o retorno de um método dentro de uma variavel

**public** **class** TestaSaldoNegativo {

**public** **static** **void** main(String[] args) {

Conta contaDaPriscilla = **new** Conta();

contaDaPriscilla.deposita(100);

System.***out***.println(contaDaPriscilla.saldo);

System.***out***.println(contaDaPriscilla.saca(200));

System.***out***.println(contaDaPriscilla.saldo);

}

}

- no codigo acima teremos a saída false impressa no console indicando a nossa regra de negocio que nenhuma conta pode ter valores negativos

- porem se acessarmos diretamente o atributo ***saldo*** e definirmos o valor 200 conseguiremos ter um valor negativo

contaDaPriscilla.saldo = contaDaPriscilla.saldo - 200;

System.***out***.println(contaDaPriscilla.saldo);

- o ideal é que utilizemos sempre os métodos e nunca diretamente nos atributos

- a ideia é utilizar uma interface adequada para melhor funcionalidade do sistema, e encapsulamos o funcionamento do mesmo

- queremos no exemplo que a manipulação direta de atributos não seja possível

- no Java podemos ocultar um atributo deixa-lo privado

- na classe ***Conta*** escrevemos ao lado do atributo que queremos encapsular a palavra-chave ***private***

**private** **double** saldo;

**int** agencia;

**int** numero;

Cliente titular;

- a partir do momento que um atributo se torna privado isso quer dizer que ele não pode ser lido ou modificado (a não ser dentro da própria classe), e esse é o conceito principal de encapsulamento

- agora não conseguimos mais imprimir o valor na classe TestaSaldoNegativo e em qualquer outra que esteja utilizando o método privado ***saldo***

- é preciso utilizar um novo método para acessar o saldo

- na classe ***Conta*** criaremos um novo método que pega e devolve o saldo chamaremos de ***retornaSaldo***

- esse novo método não recebe parâmetros mas mesmo assim os parênteses são obrigatórios

- ao lado esquerdo do método colocaremos o seu retorno (double) e public

- dentro do método diremos que ele retorna o saldo através da palavra ***return***

**public** **double** consultaSaldo() {

**return** **this**.saldo;

}

- getters e setters

- o nome ***consultaSaldo()*** poderia ter qualquer outro nome por questão de convenção alteramos para ***getSaldo()***

- com essa alteração os arquivos que utilizam o ***consultaSaldo()*** não compilarão

- para resolver isso na classe ***Conta*** selecionamos o método e escolheremos **REFACTOR 🡪 RENAME**

- feito isso o nome do método será alterado em todos os arquivos

- o nome desse tipo de método que simplesmente exibe uma informação é **GETTER**

- não se trata de um elemento de sintaxe do Java

- get não é uma palavra reservada do Java

- temos um método getter para saldo, pois esse atributo é privado e precisamos acessar a informação contida em saldo de outras classes

- no nosso caso não precisamos utilizar um **SET (SETTER)** saldo para modificar o mesmo elas serão realizadas pelo (saca, deposita e transfere)

- já no caso de ***agencia, conta*** e ***titular*** é diferente

- por enquanto não temos método para alterar esses atributos

- o ideal é que todos os atributos sejam privados (raros casos teremos método públicos)

- criamos um projeto novo ***bytebank-encapsulado***

- encapsulado é utilizado para elementos ocultos

- transformaremos nesse novo projeto todos os atributos da classe ***Conta*** em ***private***

- um atributo privado facilita a modificação e atualização do codigo com uma Classe sendo responsável por seus próprios atributos, a manutenção do sistema se torna localizada e mais simples

- para que os atributos sejam acessados fora da classe utilizaremos os **GETTERS**

- criaremos na classe ***Conta*** um método publico chamado ***getNumero()***

- que devolve um int e retorna numero

**public** **int** getNumero() {

**return** **this**.numero;

}

- teremos também um método que altere também esse numero (set)

- diferente do get que não precisamos de um parâmetro entre os parênteses, o set necessita de um parâmetro para ser passado para o método afinal queremos modificar algo (nesse caso numero da conta)

**public** **void** setNumero(**int** numero) {

**this**.numero = numero;

}

- criaremos uma nova classe para utilizar o setter e o getter (TestaGetSetter)

- criamos uma nova conta e instaciaremos através no ***new***

- para poder inserir um numero de conta precisaremos utilizar o setNumero, pois agora os atributos da conta da classe Conta são privados

- em muitos casos utiliza-se o mesmo nome do atributo para a variável

- para não confundirmos basta atentar ao this do lado esquerdo que marca a referencia ao atributo

**public** **class** TestaGetterSetter {

**public** **static** **void** main(String[] args) {

Conta contaDoFrancisco = **new** Conta();

contaDoFrancisco.setNumero(1000);

System.***out***.println(contaDoFrancisco.getNumero());

}

}

- uma das vantagens de utilizar métodos é que dentro do método podemos adicionar IFs, mensagens de erros, acessos a banco de dados, etc

- utilizamos o sysout e método getNumero() para imprimir o resultado

**public** **int** getAgencia() {

**return** agencia;

}