

Programação Orientada a Objetos

**Aula: Introdução a
Orientação a Objetos,
Paradigmas de
Programação e Linguagem
de Programação Java.**



Prof. Anderson Augusto Bosing

Introdução a Linguagem de Programação Java

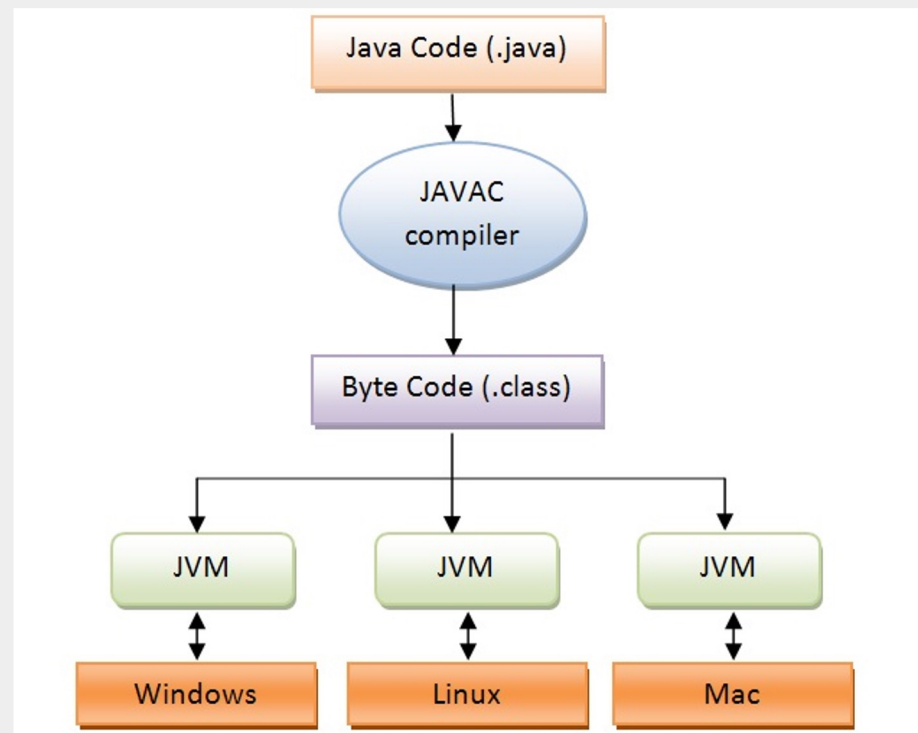
Java é uma linguagem de programação orientada a objetos que começou a ser criada em 1991, na Sun Microsystems. Teve início com o Green Project, no qual os mentores foram Patrick Naughton, Mike Sheridan, e James Gosling.

A linguagem JAVA é multiplataforma. Esta afirmação reporta ao fato de que um programa escrito na linguagem JAVA pode ser executado em qualquer plataforma (sistema operacional) sem necessidade de alterações no código-fonte.



Introdução a Linguagem de Programação Java

Tal funcionalidade é possível devido à estrutura de linguagem interpretada que caracteriza a linguagem JAVA e seu processo de compilação do código-fonte.



Introdução a Linguagem de Programação Java

Principais características e vantagens da linguagem Java:

- **Suporte à orientação a objetos;**
- **Portabilidade;**
- **Linguagem Simples;**
- **Alta Performance;**
- **Independente de plataforma;**
- **Tipada (detecta os tipos de variáveis quando declaradas);**
- **Alta aceitação de mercado tanto regional quanto mundial.**

Tipos de Dados-Java

Tipos de dados são especificados de diferentes tamanhos e valores que podem ser armazenados na variável. Existem dois tipos de dados em java:

- Tipo de dado primitivo.
- Tipo de dado não primitivo.

Tipos de Dados-Java

Dados Primitivos

- **boolean**

Exemplo de utilização

```
boolean eMenorQue = false;
```

Utilizado para armazenar valores de verdadeiro ou falso(true, false);

Tipos de Dados-Java

Dados Primitivos

➤ long

Exemplo de utilização

```
long a = 100000;
```

Utilizado para armazenar um números inteiros que não são suportados pelo tipo int.

```
16  */
17  public void main(String[] args) {
18      (Alt-Enter mostra dicas)
19      int i = 9999999999;
20
21      long l = 9999999999;
22  }
23
```

integer number too large: 9999999999

Tipos de Dados-Java

Dados Primitivos

- **double**

Exemplo de utilização

```
double i = 345.455555;
```

Utilizado para armazenar um números com casas decimais chamados de ponto-flutuante.

Tipos de Dados-Java

Dados Não Primitivos

➤ String

Exemplo de utilização

```
String i = "Professor Anderson";
```

Utilizado para armazenar valores que podem ser expressados por palavras.

Operadores de Atribuição-Java

O operador de atribuição é utilizado para definir o valor inicial ou sobrescrever o valor de uma variável.

```
int idade = 15;  
String nome = "Anderson";  
double salario = 450.00;
```

Operadores Aritméticos-Java

Os operadores aritméticos realizam as operações fundamentais da matemática entre duas variáveis e retornam o resultado.

```
int area = 2 * 2;
```

+	operador de adição
-	operador subtração
*	operador de multiplicação
/	operador de divisão
%	operador de módulo (ou resto da divisão)

Operadores de Igualdade-Java

Os operadores de igualdade verificam se o valor ou o resultado da expressão lógica à esquerda é igual (“==”) ou diferente (“!=”) ao da direita, retornando um valor booleano.

```
int valorA = 1;
int valorB = 2;

if (valorA == valorB) {
    System.out.println("Valores iguais");
} else {
    System.out.println("Valores diferentes");
}

if (valorA != valorB) {
    System.out.println("Valores diferentes");
} else {
    System.out.println("Valores iguais");
}
```

==	Utilizado quando desejamos verificar se uma variável é igual a outra.
!=	Utilizado quando desejamos verificar se uma variável é diferente de outra.

Operadores relacionais-Java

Os operadores relacionais, assim como os de igualdade, avaliam dois operandos. Neste caso, mais precisamente, definem se o operando à esquerda é menor, menor ou igual, maior ou maior ou igual ao da direita, retornando um valor booleano.

```
int valorA = 1;
int valorB = 2;

if (valorA > valorB) {
    System.out.println("maior");
}

if (valorA >= valorB) {
    System.out.println("maior ou igual");
}

if (valorA < valorB) {
    System.out.println("menor");
}

if (valorA <= valorB) {
    System.out.println("menor ou igual");
}
```

Operadores Lógicos - Java

Os operadores lógicos representam o recurso que nos permite criar expressões lógicas maiores a partir da junção de duas ou mais expressões. Para isso, aplicamos as operações lógicas E (representado por “&&”) e OU (representado por “||”).

```
if ((1 == (2 - 1)) && (2 == (1 + 1))) {  
    System.out.println("Ambas as expressões são verdadeiras");  
}
```

&&	Utilizado quando desejamos que as duas expressões sejam verdadeiras.
	Utilizado quando precisamos que pelo menos uma das expressões seja verdadeira.

Estruturas Condicionais –Java

If/Else

As estruturas condicionais possibilitam ao programa tomar decisões e alterar o seu fluxo de execução. Isso possibilita ao desenvolvedor o poder de controlar quais são as tarefas e trechos de código executados de acordo com diferentes situações, como os valores de variáveis.

```
//Se a posicao for multipla de 3
if (i % 3 == 0) {
    //Multiplos de 3
    //Índice da Posição x 30% x Valor Informado pelo Usuário
    array[i] = i * (0.3 * valueUser);
} else {
    //Índice da Posição x 10% x Valor informado pelo Usuário
    array[i] = i * (0.1 * valueUser);
}
```

```
if (somaPares.equals("par")) {
    indice = 0;
} else {
    indice = 1;
}
```

Estruturas Condicionais –Java

Switch/Case

A estrutura condicional switch/case vem como alternativa em momentos em que temos que utilizar múltiplos ifs no código. Múltiplos if/else encadeados tendem a tornar o código muito extenso, pouco legível e com baixo índice de manutenção.

```
int mes = 1;
switch (mes) {
    case 1:
        System.out.println("O mês é janeiro");
        break;
    case 2:
        System.out.println("O mês é fevereiro");
        break;
    default:
        System.out.println("Mês inválido");
        break;
}
```


Conhecendo nossa IDE



Estruturas de Repetição – Java

While

Especifica que um sistema ou programa de computador deve repetir uma instrução ou um conjunto de instruções enquanto a condição que é uma expressão booleana permanece verdadeira.

```
int num = 0;

while (num <= 99) {
    System.out.println("nmum = "+num);

    num++;
}
```

```
--- exec-maven-plugin:1.2.1:exec (default-cli) @ ListaVetores ---
nmum = 0
nmum = 1
nmum = 2
nmum = 3
nmum = 4
nmum = 5
nmum = 6
nmum = 7
nmum = 8
nmum = 9
nmum = 10
```

Estruturas de Repetição – Java

Do - While

```
int num = 0;

do {
    System.out.println("num = "+num);
    num++;
} while (num < 100);
```

```
--- exec-maven-plugin:1.2.1:exec (default-cli) @ ListaVetores ---
nnum = 0
nnum = 1
nnum = 2
nnum = 3
nnum = 4
nnum = 5
nnum = 6
nnum = 7
nnum = 8
nnum = 9
nnum = 10
```

Estruturas de Repetição – Java

For

O for também é uma instrução de repetição que processa uma instrução ou grupo de instrução zero ou mais vezes. O que muda é a sintaxe, no caso do for a variável de controle, o valor inicial, o incremento e a condição de continuação do loop, são declarados como em uma espécie de cabeçalho.

```
for (int num = 0; num < 10; num++) {  
    System.out.println("num =" + num);  
}
```

System.out.print – Java

O for também é uma instrução de repetição que processa uma instrução ou grupo de instrução zero ou mais vezes. O que muda é a sintaxe, no caso do for a variável de controle, o valor inicial, o incremento e a condição de continuação do loop, são declarados como em uma espécie de cabeçalho.

```
public static void main(String[] args) {  
    System.out.println("Bem vindos ao segundo ano da UNIPAR");  
}
```

```
--- exec-maven-plugin:1.2.1:exec (default-cli) @ ListaVetores ---  
Bem vindos ao segundo ano da UNIPAR  
-----  
BUILD SUCCESS
```



Classe Scanner – Java

Como ler dados informados pelo usuário ?

A classe Scanner apareceu a partir do Java 5 com o objetivo de facilitar a entrada de dados no modo Console. Uma das características mais interessante da classe Scanner é a possibilidade de obter o valor digitado diretamente no formato do tipo primitivo que o usuário digitar. Para isso basta utilizarmos os métodos next do tipo primitivo no formato nextTipoDado().

```
Scanner scanner = new Scanner(System.in);  
System.out.print("Qual o seu nome: ");  
String nome = scanner.next();  
System.out.println("Seja bem vindo " + nome + "!");
```

```
Qual o seu nome: Anderson  
Seja bem vindo Anderson!  
-----  
BUILD SUCCESS
```

Variáveis em Java

```
String nome = "Professor Anderson";  
int idade;  
int Idade;  
idade = 29;
```

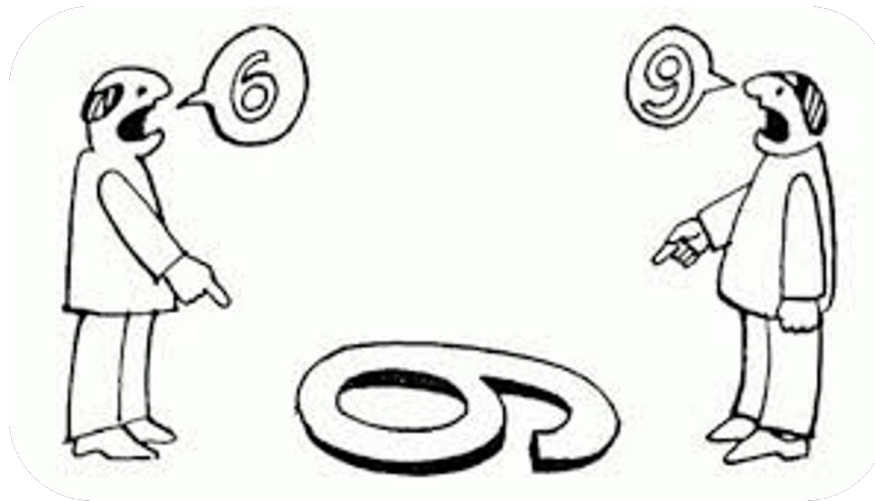
Programação Orientada a Objetos

Aula: Introdução a Classes e Objetos.



Prof. Anderson Augusto Bosing

O que é um **Paradigma**?



O que é um **Paradigma**?

Um exemplo que serve como modelo ou padrão.

O que são Paradigmas da Programação?

O que são Paradigmas da Programação?

Um paradigma é um estilo de programação, um modelo, uma metodologia.

Não se trata de uma linguagem, mas a forma como você soluciona problemas usando uma determinada linguagem de programação.

Paradigmas da Programação

Procedural

Consiste em um modelo de programação que funciona como uma espécie de lista de instruções que são executadas em forma de passo a passo.

Ex:

C

Pascal

Paradigmas da Programação

POO

Este paradigma é o que mais reflete os problemas atuais. Um programa OO consistem em objetos que enviam mensagens uns para os outros. Estes objetos no programa correspondem diretamente a objetos atuais, tais como pessoas, máquinas, departamentos, documentos e assim por diante.

Ex:

Java

C#



O paradigma de Orientação a Objetos

Historicamente antes de 1975 a maioria das empresa de software não usava nenhuma técnica específica, cada indivíduo trabalhava do seu próprio jeito.

Grandes avanços foram feitos aproximadamente entre 1975 e 1985, com o desenvolvimento assim chamado paradigma clássico ou estruturado. Essa abordagem parecia extremamente promissora para a época. Todavia à medida que o tempo foi passando, constatou-se que ela ficou aquém do esperado em dois principais aspectos:

1. Algumas vezes o paradigma e suas técnicas eram incapazes de lidar com o tamanho cada vez maior dos produtos de software, isso é, as técnicas clássicas eram adequadas para elaborar produtos com escala pequena ou média.
2. O paradigma clássico não estava a altura das expectativas iniciais durante a manutenção pós-entrega.

O paradigma de Orientação a Objetos

A principal razão para o sucesso limitado desse paradigma clássico foi que as técnicas são orientadas a operações ou a atributos(dados), mas não a ambos ao mesmo tempo.

Em contraste, o paradigma de orientação a objetos considera igualmente importantes tanto os atributos quanto as operações. Uma maneira simplista de compreender um objeto é vê-lo como um artefato de software unificado, que incorpora tanto os atributos quanto as operações realizadas sobre os atributos

Evolução Histórica

Bezerra (2015) apresenta um breve resumo histórico da evolução das técnicas de desenvolvimento, para explicar como chegamos ao cenário atual.

Décadas de 1950/1960: Os sistemas eram bem simples, e o seu desenvolvimento era direto ao assunto, não tinha um planejamento inicial, como dizem, “ad hoc”. Como os sistemas eram significativamente mais simples, as técnicas de modelagem também: Eram usados fluxogramas e diagramas de módulos.

Década de 1970: Começaram a surgir computadores mais avançados e acessíveis. Houve grande ampliação do mercado computacional. Sistemas mais complexos começavam a surgir. Consequentemente, modelos mais robustos foram propostos. Os autores Larry Constantine e Edward Yourdon são grandes colaboradores nessas técnicas.

Evolução Histórica

Década de 1980: Nessa fase os computadores se tornaram ainda mais avançados e mais baratos. Surge a necessidade por interfaces mais sofisticadas, o que originou a produção de sistemas de softwares mais complexos. A Análise Estruturada surgiu no início desse período, com os trabalhos de Edward Yourdon, Peter Coad, Tom De Marco, James Martin e Chris Gane.

Década de 1990: No início dessa década é o período em que surge um novo paradigma de modelagem, a Análise Orientada a Objetos, como resposta a dificuldades encontradas na aplicação da análise estruturada em algumas aplicações. Grandes colaboradores desse paradigma são Sally Shlaer, Stephen Mellor, Rebecca Wirfs-Brock, James Rumbaugh, Grady Booch e Ivar Jacobson. Já no fim da década, o paradigma da orientação a objetos atinge sua maturidade. Os conceitos de padrões de projeto, frameworks, componentes e qualidade começam a ganhar espaço. Surge a Linguagem de Modelagem Unificada UML.



Evolução Histórica

Década de 2000: O reúso por meio de padrões de projetos e frameworks se solidifica. As denominadas metodologias ágeis começam a ganhar espaço. Técnicas de testes automatizados e refatoração começaram a se difundir entre os desenvolvedores que trabalham com orientação a objeto. Grandes nomes dessa fase são: Rebecca Wirfs-Brock, Martin Fowler e Eric Vans.

Como percebemos, durante a década de 90 surgiram várias propostas e técnicas para modelagem de sistema segundo o paradigma da orientação a objeto. Era comum, durante essa década, duas técnicas possuírem diferentes notações gráficas para modelar a mesma perspectiva de um sistema. Todas tinham pontos fortes e fracos em relação à notação utilizada, mas via-se a necessidade de uma notação que viesse a se tornar um padrão para a modelagem de sistemas orientados a objeto, e que fosse amplamente aceita, nas indústrias e na academia.

Alguns esforços surgiram em 1996 para essa padronização, o que resultou na definição da UML (Unified Modeling Language), que detalharemos a seguir em um tópico específico, mas antes falaremos na Modelagem Estruturada e da Modelagem Orientada a Objetos.



Evolução Histórica

Pode-se construir uma casa para um cachorro apenas juntando algumas tábuas, alguns pregos e algumas ferramentas básicas. Com um planejamento mínimo, em poucas horas e com um pouco de habilidade, nosso cachorro terá uma casa (BOOCH et al., 2000).

Introdução à Tecnologia de Objetos

Hoje, como a demanda por software novo e mais poderoso está aumentando, construir softwares de maneira rápida, correta e econômica continua a ser um objetivo indefinido.

Objetos ou, mais precisamente, as classes de onde os objetos são essencialmente componentes reutilizáveis de software.

Há objetos data, objetos data/hora, objetos áudio, objetos vídeo, objetos automóvel, objetos pessoas etc.

Quase qualquer substantivo pode ser razoavelmente representado como um objeto de software em termos dos **atributos** (por exemplo, nome, cor e tamanho) e **comportamentos** (por exemplo, calcular, mover e comunicar).



Introdução à Tecnologia de Objetos

Quando programamos estamos modelando aspectos do ‘mundo real’ utilizando uma linguagem de programação. Assim sempre temos um aspecto do ‘mundo real’, a representação deste aspecto no ‘mundo computacional’ (em tempo de execução) e a descrição deste aspecto no ‘mundo linguístico’ (em uma linguagem de programação).

O automóvel como um objeto

Vamos supor que você queira dirigir um carro e fazê-lo andar mais rápido pisando no pedal acelerador. O que deve acontecer antes que você possa fazer isso?

O automóvel como um objeto

Vamos supor que você queira dirigir um carro e fazê-lo andar mais rápido pisando no pedal acelerador. O que deve acontecer antes que você possa fazer isso?

Alguém precisa projetá-lo.

Criar uma especificação que vai servir como base para fabricação dos carros que vão ser utilizados nas ruas.

Considerando um software para um banco.

O que toda conta corrente tem que é importante para nós?

Considerando um software para um banco.

O que toda conta corrente tem que é importante para nós?

- Número da conta
- Nome do titular da conta
- Saldo
- Limite

Considerando um software para um banco.

O que toda conta corrente faz que é importante para nós?

O que pedimos à conta corrente?

Considerando um software para um banco.

O que toda conta corrente faz que é importante para nós?

O que pedimos à conta corrente?

- Saca uma quantidade x ;
- Deposita uma quantidade x ;
- Consultar o saldo atual;
- Imprimir extrato.
- Transfere uma quantidade x para uma outra conta y ;

Considerando um software para um banco.

De acordo com o que levantamos então temos uma especificação de uma conta. Sendo ela:

Atributos de Uma conta

- Número da conta
- Nome do titular da conta
- Saldo
- Limite

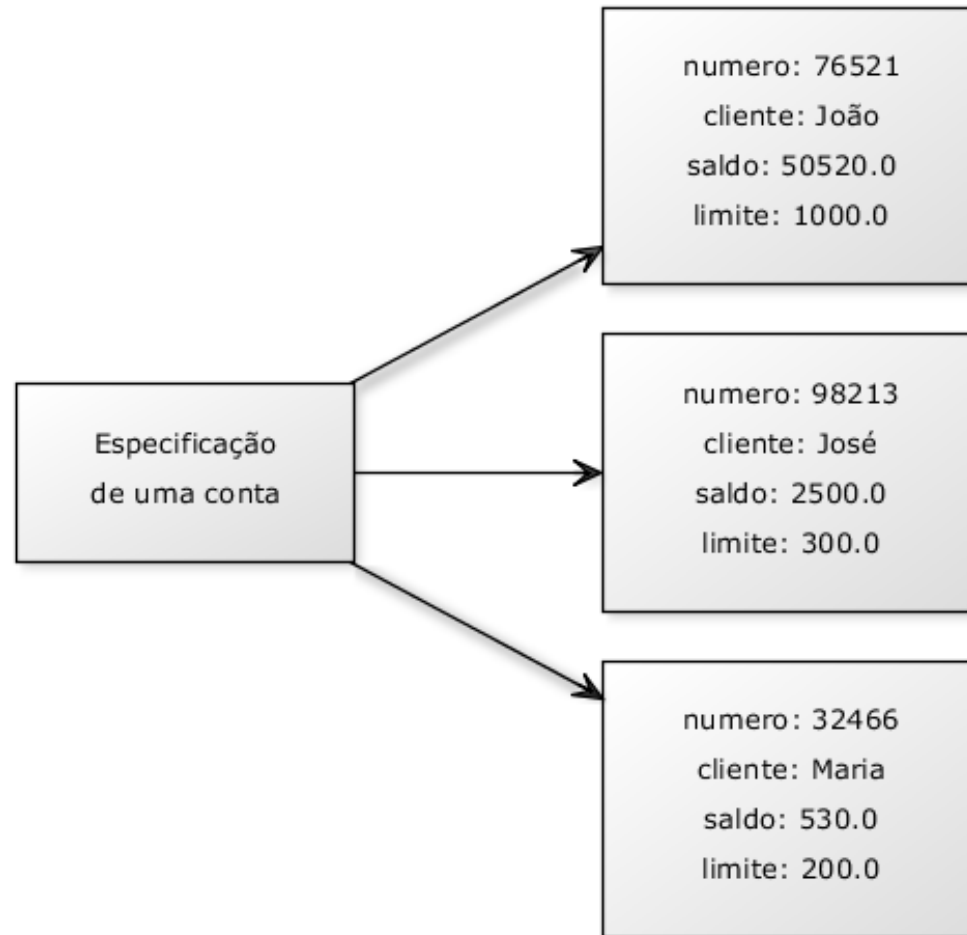
Métodos de uma Conta

- Saca uma quantidade x ;
- Deposita uma quantidade x ;
- Consultar o saldo atual;
- Imprimir extrato.
- Transfere uma quantidade x para uma outra conta y ;

Mas o que temos ainda é o projeto dela, antes de a utilizarmos precisamos construir uma conta.



Considerando um software para um banco.



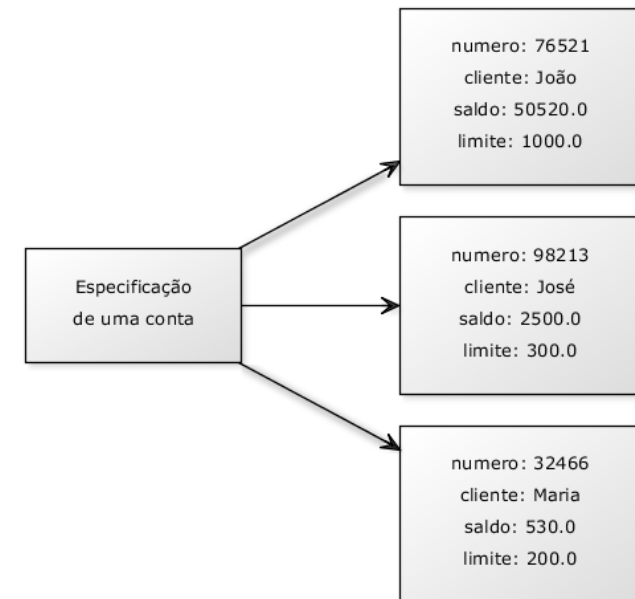
Considerando um software para um banco.

Na figura anterior podemos observar que ao lado esquerdo temos a especificação de uma conta, mas essa especificação é uma conta ?

Nós depositamos e sacamos dinheiro desse papel?

Não. Utilizamos a especificação da Conta para poder criar instâncias que realmente são contas, nas quais podemos realizar as operações que criamos.

Apesar de declararmos que toda conta tem um saldo, um número e uma agência no pedaço de, são nas instâncias desse projeto em que realmente há espaço para armazenar esses valores.

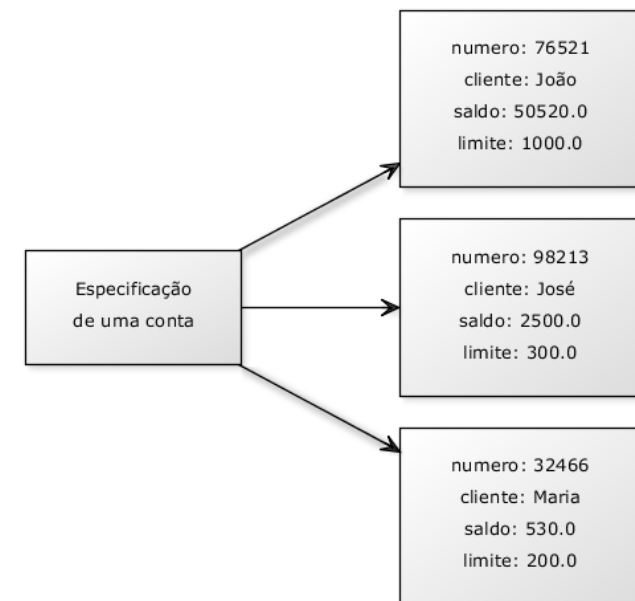


Considerando um software para um banco.

Ao projeto da conta, isto é, à especificação da conta, damos o nome de **classe**. Ao que podemos construir a partir desse projeto que são as contas de verdade, damos o nome de **objetos**.

As características da conta damos o nome de **atributos**(Numero, Saldo, Cliente).

E aos comportamentos desta conta damos o nome de **métodos**(Sacar, depositar, imprimir extrato).



Um outro exemplo: uma receita de bolo

A pergunta é certa: você come uma receita de bolo? Não.

Precisamos **instanciá-la** e fazer um **objeto bolo** a partir dessa **especificação (a classe)** para utilizá-la.

Podemos criar centenas de bolos com base nessa **classe (a receita, no caso)**. Eles podem ser bem semelhantes, alguns até idênticos, mas são **objetos diferentes**.

Um outro exemplo: planta de uma casa

A planta de uma casa é uma casa? Definitivamente, não.

Não podemos morar dentro da planta de uma casa nem podemos abrir sua porta ou pintar suas paredes.

Precisamos, antes, construir instâncias a partir dessa planta. Essas instâncias, sim, podemos pintar, decorar ou morar dentro.

Definições Técnicas

Classe

Assim como os objetos do mundo real, uma classe agrupa os objetos pelos seus comportamentos e atributos comuns.

Uma classe define os atributos e comportamentos comuns compartilhados por uma tipo de objeto. Os objetos de certo tipo ou classificação compartilham os mesmos comportamentos e atributos.

Definições Técnicas

Objeto

Um objeto é uma instância de uma classe.

Atributo

Atributos são as características de uma classe visíveis externamente. A cor de um carro e o seu modelo são exemplos de atributos.

Método

Métodos definem as habilidades dos objetos

Vamos codificar.

- Criar uma classe câmera.



- Criar uma classe gato.



Vamos Praticar?

- Lista de Exercícios em Orientação a Objetos com Java disponível no classroom.



Programação Orientada a Objetos

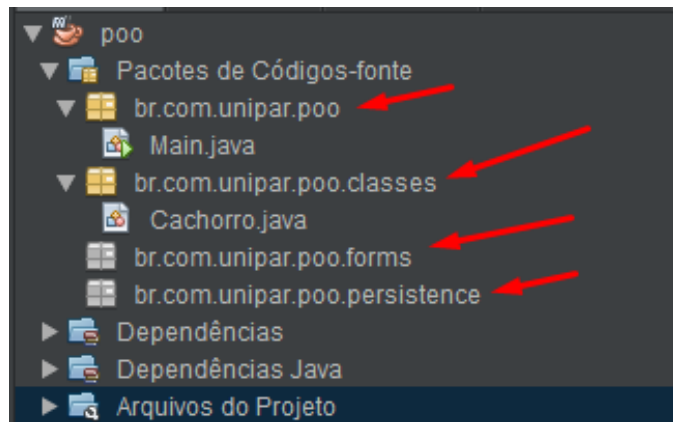
**Aula: Modificadores de acesso,
Métodos Acessores.**



Prof. Anderson Augusto Bosing

Pacotes

Pacotes java são utilizados para organizar as classes da sua aplicação. Um programa pode, facilmente, ter mais de centenas de classes. Então é muito importante que todos os seus componentes fiquem organizados. Podemos pensar nos pacotes como uma pasta do seu sistema de arquivos.



Documents > NetBeansProjects > poo > src > main > java > br > com > unipar > poo >

Nome	Data de modificação	Tipo	Tamanho
classes	05/04/2022 18:30	Pasta de arquivos	
forms	05/04/2022 18:30	Pasta de arquivos	
persistence	05/04/2022 18:30	Pasta de arquivos	
Main.java	05/04/2022 18:29	Arquivo Fonte Java	1 KB

Modificadores de Acesso a Nível de Classe - public

Modificador public torna uma classe visível:

Para qualquer outra classe e em qualquer pacote.

```
*/  
public class Cachorro {  
    |  
}
```

Modificadores de Acesso a Nível de Classe - default

Modificador vazio ou default torna uma classe visível:

Torna uma classe visível apenas para classes do mesmo pacote.

```
*/  
class Cachorro {  
  
}
```

Modificadores de Acesso a Nível de Atributos e Métodos(Membros) - public

public torna um membro acessível:

Em qualquer lugar e a qualquer outra classe que possa visualizar a classe que contém o membro.

```
public class Cachorro {  
    public String nome;  
}
```

```
public class Main {  
    /**  
     * @param args the command line arguments  
     */  
    public static void main(String[] args) {  
        Cachorro dog = new Cachorro();  
        dog.nome = "bob";  
    }  
}
```

Modificadores de Acesso a Nível de Atributos e Métodos(Membros) - **protected**

protected torna um membro acessível às classes:

Do mesmo pacote.

Os membros herdados não são acessíveis a outras classes fora do pacote em que foram declarados.

```
*/  
public class Cachorro {  
    protected String nome;  
}
```

```
*/  
public static void main(String[] args) {  
    Cachorro dog = new Cachorro();  
    dog.nome = "bob";  
}
```

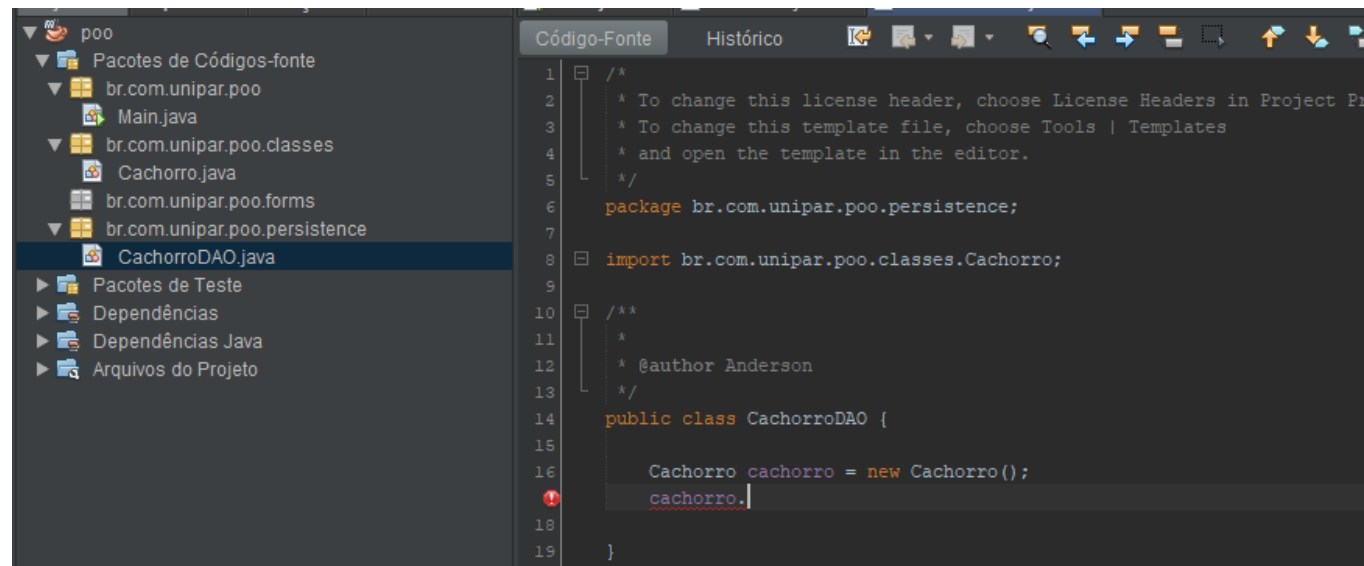
```
*/  
public sta  
Cachorro  
dog.nome = "bob";
```

nome has protected access in Cachorro
(Alt-Enter mostra dicas)

Modificadores de Acesso a Nível de Atributos e Métodos(Membros) - default

default (sem modificador explícito) torna um membro acessível apenas para classes do mesmo pacote.

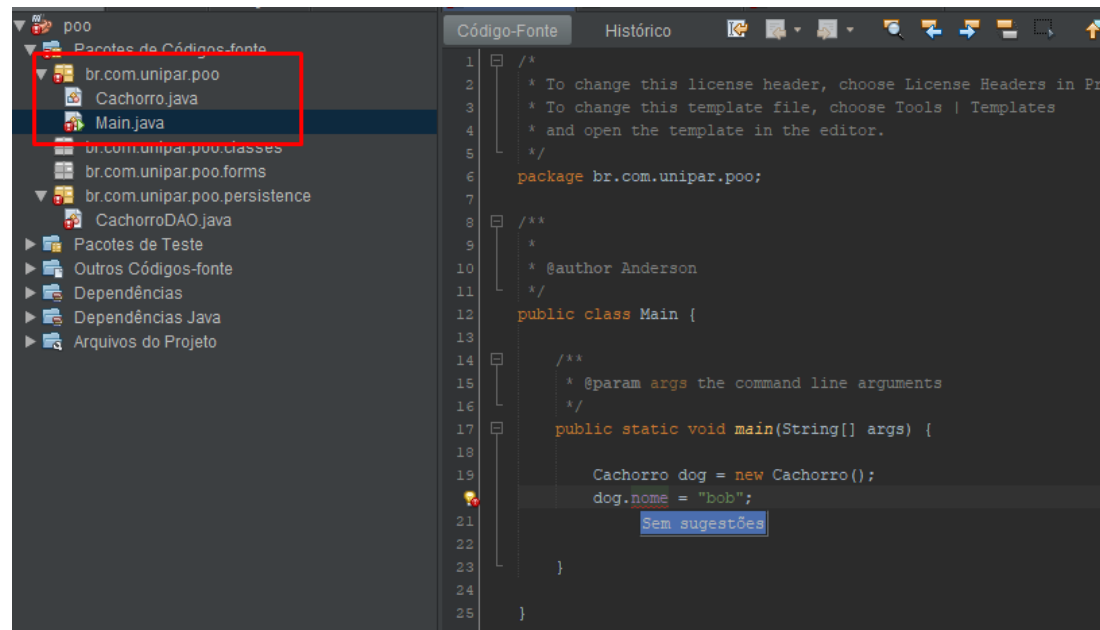
```
public class Cachorro {  
    String nome;  
}
```



Modificadores de Acesso a Nível de Atributos e Métodos(Membros) - private

private torna um membro acessível apenas para a classe que o contém.

```
public class Cachorro {  
    private String nome;  
}
```



Objetos não devem alterar os estados dos outros ou seus atributos.

Exemplos:

Pessoa e um Carro.

Pessoa e um Porta.



Professor, mas se um objeto não pode alterar diretamente os atributos e o estado de outro objeto, como fazemos nosso código abaixo?

```
public class Cachorro {  
    String nome;  
}
```

```
public class Main {  
    /**  
     * @param args the command line arguments  
     */  
    public static void main(String[] args) {  
        Cachorro dog = new Cachorro();  
        dog.nome = "bob";  
    }  
}
```


Professor, mas se um objeto não pode alterar diretamente os atributos e o estado de outro objeto, como fazemos nosso código abaixo?

Através dos métodos acessores.

Métodos Acessores(Get e Set)

Também conhecidos como getter e setters, são métodos utilizados para que seja possível acessar e modificar os valores de variáveis com modificador de acesso private.

Getters

O propósito de um getter é obter o valor de uma variável declarada como `private` e permitir sua leitura a partir de outra classe.

Setters

Um setter é um método que permite modificar o valor de um atributo da classe que não seja acessível diretamente por ser privado.

Getters e Setters

```
1  */
2  public class Cachorro {
3      //Atributo privado
4      private String nome;
5
6      //No caso do setter o retorno do metodo é void ou seja sem retorno pois
7      //usamos o metodo para setar o valor a um atributo privado
8      //Acessor do metodo + Tipo de Retorno + nomeMetodo(Tipo e Parametro de Entrada)
9      public void setNome(String nome) {
10         this.nome = nome; //this identifica o contexto do que está sendo setado
11                             //nesse caso o nosso contexto é a própria classe
12                             //fazendo com que this.nome seja diferente da variavel de entrada nome
13     }
14
15     //No caso do getter como apenas queremos buscar o valor do atributo privado não temos
16     //parametros de entrada no metodo mas temos o tipo de retorno
17     //nesse caso como o atributo se trata de uma string
18     //o tipo de retorno é uma string
19     //Acessor do metodo + Tipo de Retorno + nomeMetodo()
20     public String getNome() {
21         return nome; //return é um comando reservado que identifica no java qual será o retorno do metodo
22     }
23
24 }
25
```

Vamos codificar.

- Criar uma classe câmera.



- Criar uma classe gato.



Vamos codificar.

