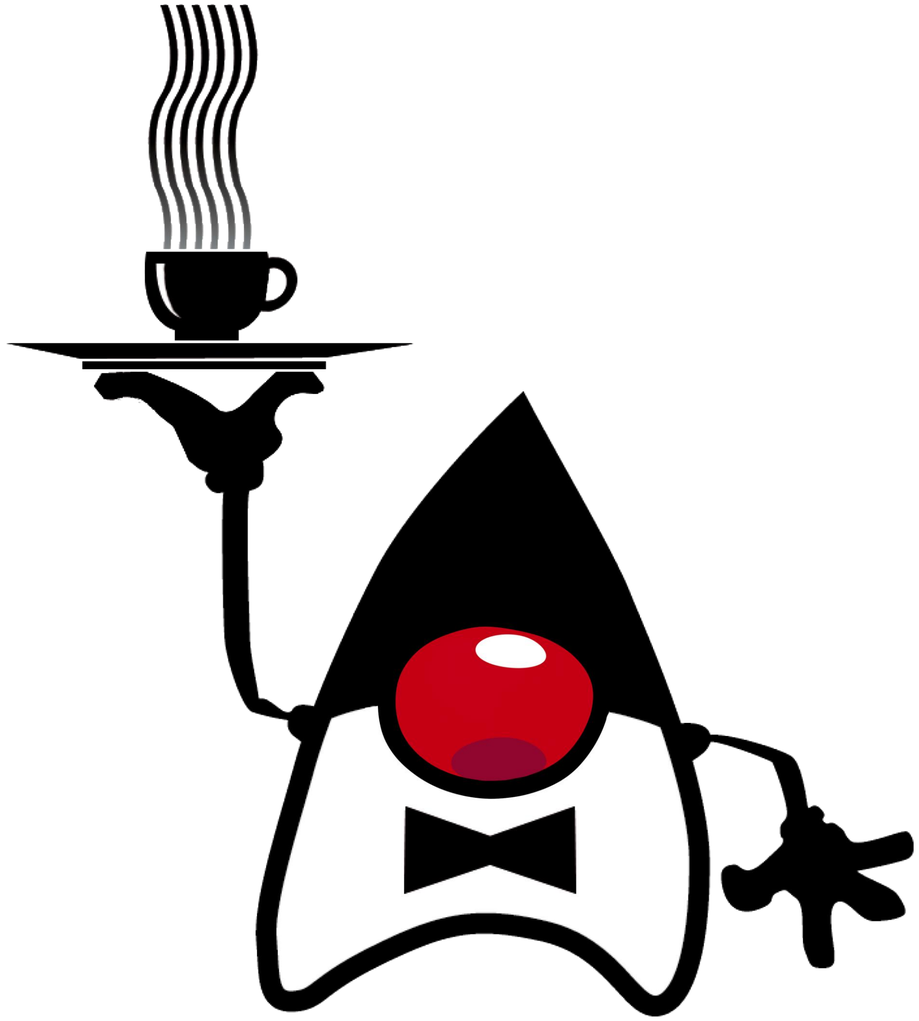
Trilha Java

Encontro 15 – (POO)
Visibilidade e Métodos Especiais



vnt/school
powered by venturus

Recapitulação

1. Introdução
2. Objetos
3. Classes
4. Criando classes e objetos
5. Abstração

vnt/school
powered by venturus

Agenda

1. UML.
2. Visibilidade.

   Atributos e Métodos
3. Métodos Especiais.

   Get, Set, Construtor
4. Exercícios.

vnt/school
powered by venturus

UML

Linguagem de Modelagem Unificada

vnt /school

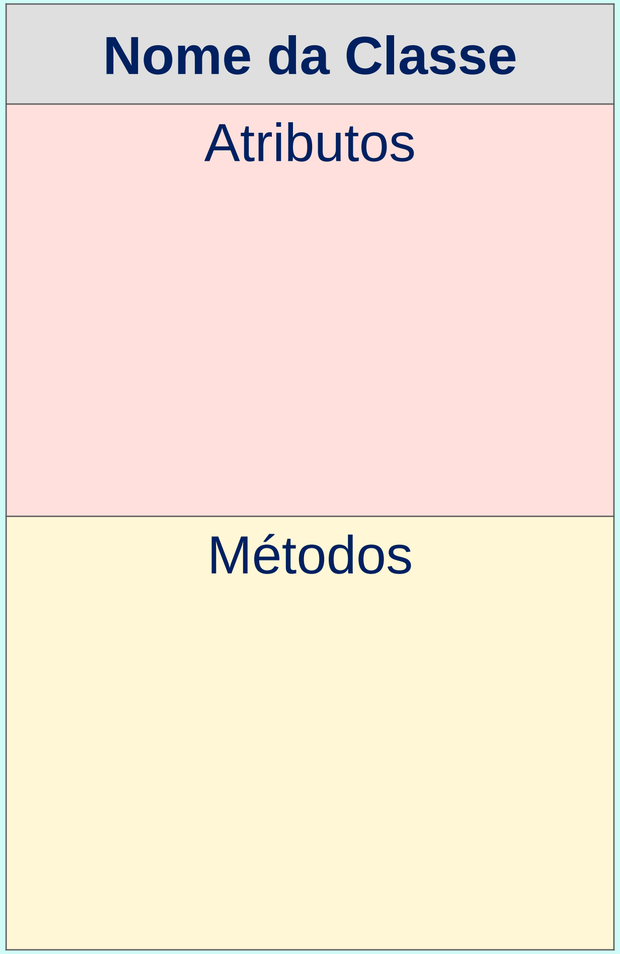
powered by venturus

UML

Do inglês UML (**Unified Modeling Language**)

Importante para POO.

Em especial, vamos trabalhar com o Diagrama de Classes.



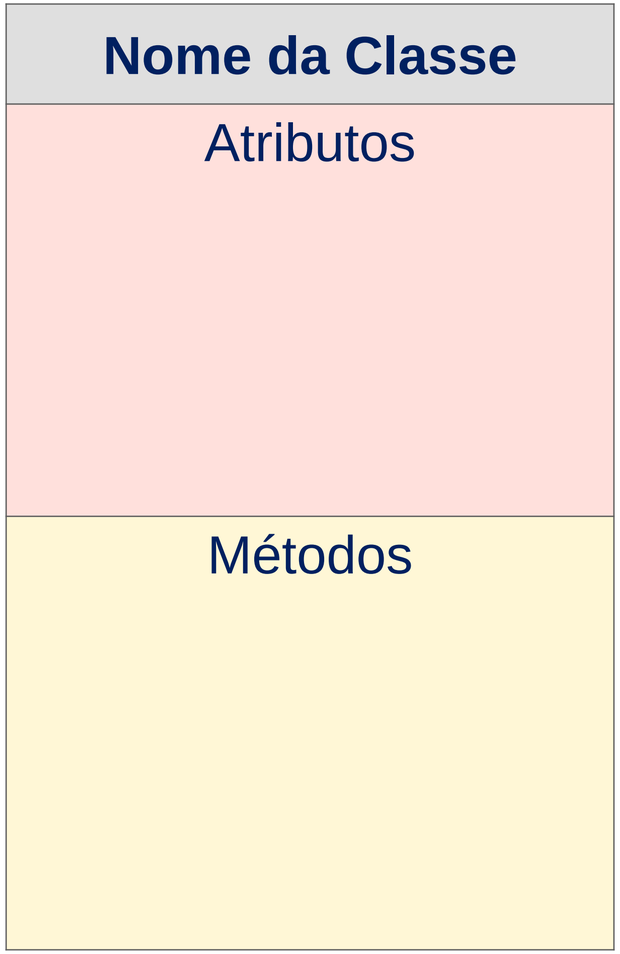
vnt/school
powered by venturus

UML

Pra que serve o diagrama de classes?

Em diagramas de classes, toda classe é representada por um retângulo.

E dentro dessa classe podemos descrever os métodos e atributos que estarão presentes.



UML

A classe é escrita com letra inicial maiúscula.

Os atributos e métodos são escritos com letra inicial minúscula.

Os atributos e métodos são separados no diagrama.

Caneta
modelo cor ponta carga tampada
escrever() rabiscar() pintar() tampar() destampar()

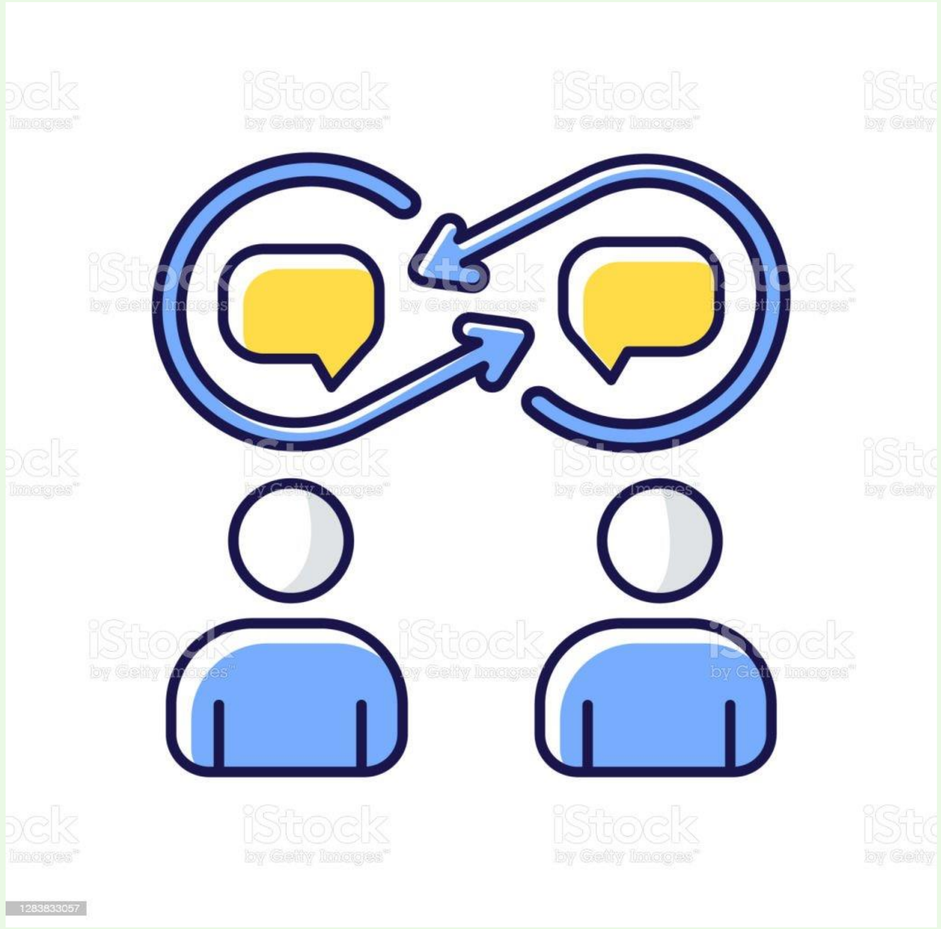
vnt /school
powered by venturus

UML

Ainda pelo diagrama é possível tratar a questão de visibilidade dos atributos e métodos.

É possível tratar do relacionamento entre classes.

Tópicos que serão discutidos ao longo do curso. (Final da aula – Dica de Hoje)



vnt/school
powered by venturus

Visibilidade

Atributos e Métodos

vnt /school

powered by venturus

Visibilidade

Modificadores de Visibilidade:

- **Público (+),**
- **Privado (-),**
- **Protegido (#)**

Indica o nível de acesso aos componentes internos de uma classe.

Quais são os componentes internos?

Atributos e Métodos.

Visibilidade

(+) Publico

A classe atual e todas as outras classes podem ter acesso aos atributos e métodos.

(-) Privado

Somente a classe atual pode ter acesso aos atributos e métodos.

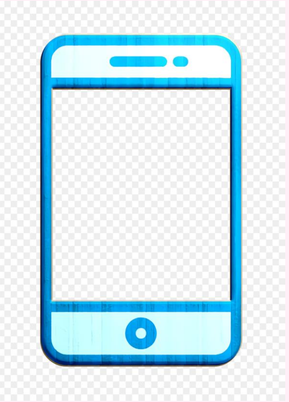
(#) Protegido

A classe atual e todas as suas subclasses podem ter acesso aos atributos e métodos.

Público

Protegido

Privado



vnt/school

powered by venturus

Visibilidade

O diagrama de classe pode ser representado conforme a figura ao lado.

Veja que neste caso acrescentamos a visibilidade.

Como fazer com que essa classe funcione efetivamente?

Visibilidade

Basta escrever a palavra Público, Privado ou Protegido na frente dos atributos e métodos.

Conforme a visibilidade de cada um.

```
Classe Caneta
publico modelo: caractere
publico cor: caractere
privado ponta: real
protegido carga: inteiro
protegido tampada: logico
publico Metodo rabiscar()
...
fimMetodo
publico Metodo tampar()
...
fimMetodo
FimClasse
```

vnt/school
powered by venturus

Visibilidade

```
c1 = new Caneta
c1.modelo = "BIC"
c1.cor = "Azul"
c1.ponta = 0.5
c1.carga = 90
c1.tampada = falso
c1.rabiscar( )
c1.rabiscar( )
```

Qual o erro
na construção
desse objeto?

```
Classe Caneta
publico modelo: caractere
publico cor: caractere
privado ponta: real
protegido carga: inteiro
protegido tampada: logico
publico Metodo rabiscar()
...
fimMetodo
publico Metodo tampar()
...
fimMetodo
FimClasse
```

vnt/school
powered by venturus

Visibilidade

Será que temos acesso a tudo que tem no controle de tv?

Será que precisamos ter todo acesso?

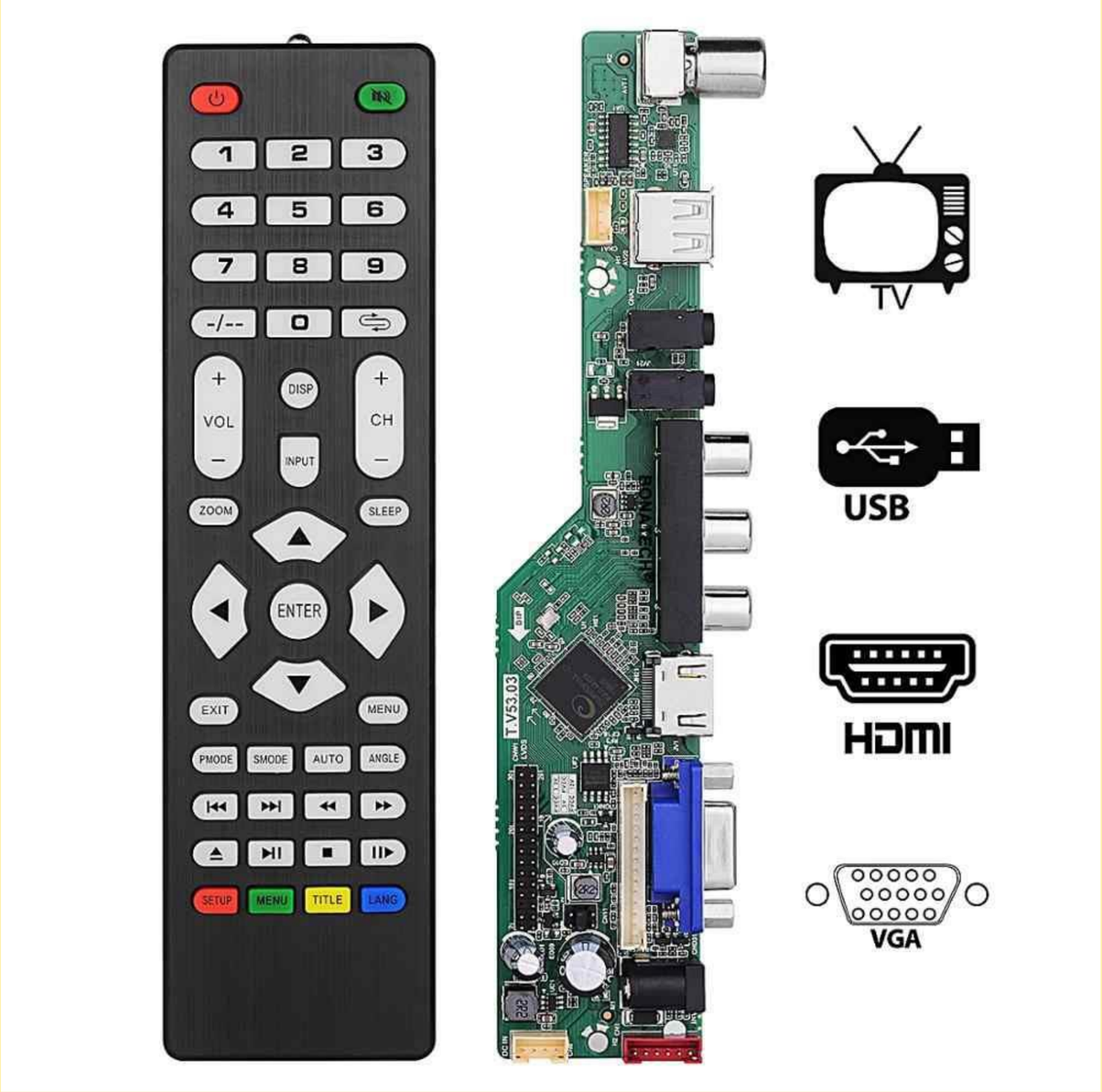
É importante limitar os acessos?

vnt/school
powered by venturus

Visibilidade

Veja os detalhes por dentro do controle. Algumas partes dele, o público **não** pode **acessar**.

Isso é importante em Java, pois os **modificadores** permitem "**proteger**" o acesso a um atributo, método ou até mesmo uma classe.



vnt/school
powered by venturus

Visibilidade

Modificar o modelo, a cor e a carga é possível. **Publico.**

Impossível modificar a ponta. **Privado.**

```
package aula15;
public class Exemplo1 {
    public static void main(String[] args) {
        Caneta c1 = new Caneta();
        c1.cor = "Azul";
        c1.modelo = "Bic";
        c1.carga = 90;
        c1.ponta = 0.7f;
        c1.tampada = false;
        c1.rabiscar();
        c1.status();
        System.out.println(" ");
```

vnt/school
powered by venturus

Visibilidade

Para visibilidade **protegido:** Porque é possível modificar o objeto c1 ?

Neste caso, veja que o método “**main**” está dentro da classe **Exemplo1**.

Isto possibilita o acesso de mudança.

```
package aula15;
public class Exemplo1 {
    public static void main(String[] args) {
        Caneta c1 = new Caneta();
        c1.cor = "Azul";
        c1.modelo = "Bic";
        c1.carga = 90;
        c1.ponta = 0.7f;
        c1.tampada = false;
        c1.rabiscar();
        c1.status();
        System.out.println(" ");
```

vnt/school
powered by venturus

Visibilidade

O **método** segue regras semelhantes.

Se o **método** rabiscar fosse colocado como Privado, o acesso seria negado.

```
package aula15;
public class Exemplo1 {
    public static void main(String[] args) {
        Caneta c1 = new Caneta();
        c1.cor = "Azul";
        c1.modelo = "Bic";
        c1.carga = 90;
        c1.ponta = 0.7f;
        c1.tampada = false;
        c1.rabiscar();
        c1.status();
        System.out.println(" ");
```

Visibilidade

Faça o atributo “**tampada**” ser “Privado”.

Em seguida, faça os métodos “**tampar**” e “**destampar**” se tornarem “Público”.

Será possível modificar o objeto c1?

vnt/school
powered by venturus

Classe Caneta
publico modelo: caractere
publico cor: caractere
privado ponta: real
protegido carga: inteiro
privado tampada: logico
publico Metodo destampar()
...
fimMetodo
publico Metodo tampar()
...
fimMetodo
FimClasse

Visibilidade

Sim, é possível!!

O método “**tampar**” e “**destampar**” é **público** e isso permite alterar no atributo “tampada” que é privado.

Perceba que a **alteração** do atributo “tampada” é feito por um **outro caminho**.

vnt/school
powered by venturus

Classe Caneta
publico modelo: caractere
publico cor: caractere
privado ponta: real
protegido carga: inteiro
privado tampada: logico
publico Metodo destampar()
...
fimMetodo
publico Metodo tampar()
...
fimMetodo
FimClasse

Visibilidade

Suponha que você compre um produto no supermercado que custa R$ 40,00.

Ao passar pelo “**caixa**”, você dar ao “**atendente**” a nota de R$ 50,00.

Perceba que o seu **acesso** ao “**caixa**” é **privado**. Mas o seu acesso ao “**atendente**” é **publico**.

O “**atendente**” consegue acessar o “**caixa**”, depositar os R$ 50,00 e retirar R$ 10,00 para te dar de troco.

Ele consegue fazer a alteração do “caixa”.

vnt /school
powered by venturus

Exercício

Visibilidade

vnt /school

powered by venturus

Atividade 1

Fazer um programa para ler os dados de um produto em estoque (nome, preço e quantidade no estoque). Em seguida mostrar os dados do produto (nome, preço, quantidade no estoque, valor total no estoque).

Realizar uma entrada no estoque e mostrar novamente os dados do produto atualizado.

Realizar uma saída no estoque e mostrar novamente os dados do produto atualizado.

Para resolver este problema, você deve criar uma CLASSE conforme projeto ao lado.

Produto
+ nome: String # preco: double # quantidade: int
+ totalValorEmEstoque(): double + addProdutos(): void + removeProdutos(): void

vnt/school
powered by venturus

Atividade 1

```java
package aula15;
public class Produto {
    public String nome;
    protected double preco;
    protected int quantidade;
    public double totalValorEmEstoque() {
        return preco * quantidade;
    }
    public void addProdutos(int quantidade) {
        this.quantidade += quantidade;
    }
    public void removeProdutos(int quantidade) {
        this.quantidade -= quantidade;
    }
```

??

vnt/school
powered by venturus

Coffee time!

Métodos Especiais

Get, Set e Construtor

vnt/school
powered by venturus

Métodos Especiais

Imagine que seja criado o objeto:

e = new Escritório

Consequentemente, pode-se afirmar que uma classe Escritório já foi criada.

vnt/school
powered by venturus

Métodos Especiais

Suponha ser um **escritório** de contabilidade que administra documentos de **várias pessoas**.

Imagine que uma certo homem queira **acessar** o **Escritório** para saber quantos documentos dele estão ali.

vnt/school
powered by venturus

Métodos Especiais

Portanto, ele faz uma consulta ao atributo **totalDoc**.

E descobre que tem 5 documentos.

Em seguida, uma mulher também tenta acessar o Escritório pra descobrir quantos documentos dela estão ali.

Ela descobre que existem 12 documentos seus, no Escritório.

```
e = new Escritório
t = e.totalDoc
```

vnt/school
powered by venturus

Métodos Especiais

Perceba que o escritório deve ter **vários clientes**. Logo, se todos tentarem acessar por conta própria todos os documentos, isso pode levar a sérios problemas.

É **importante**, ter mais **segurança** no processo.

vnt/school
powered by venturus

Métodos Especiais

Para executar tais funções usamos os métodos Getters e Setters.

- **Getters** – “pegar” algo, acessar algo.

- **Setters** – “colocar” algo, modificar algo.

vnt/school
powered by venturus

Métodos Especiais: Get

É importante colocar entre o rapaz e o Escritório, uma pessoa para controlar esses acessos.

Isso seria o **método get**.

vnt/school
powered by venturus

Métodos Especiais: Get

Portanto, o rapaz pergunta para a pessoa autorizada quantos documentos ele tem.

A pessoa autorizada acessa o Escritório e em seguida retorna para o rapaz a informação de 5 documentos.

Perceba que existe um método chamado **getTotalDoc()**.

```
e = new Escritório
t = e.getTtotalDoc()
```

vnt/school
powered by venturus

Métodos Especiais: Get

Muitos acabam argumentando que esse procedimento é mais complicado. O que pode ser verdade!!

Qual o benefício?

Muito mais seguro. Não dar acesso direto ao objeto.

Essa proteção é extremamente importante. E você consegue dar acesso ao atributo sem dar acesso direto ao atributo.

Métodos Especiais: Get

Comparando os dois!!

São muito parecidos.

Mas o segundo caso apresenta muito mais segurança.

```
e = new Escritório
t = e.totalDoc
```

```
e = new Escritório
t = e.getTtotalDoc()
```

vnt/school
powered by venturus

Métodos Especiais: Set

Suponha que agora tanto o rapaz como a moça desejam adicionar um documento ao Escritório.

Veja que o atributo **totalDoc** é adicionado de **+1**.

Existe segurança no processo?

e = new Escritório
e.totalDoc = e.totalDoc + 1

vnt/school
powered by venturus

Métodos Especiais: Set

O método **setTotDoc()** também representa a pessoa autorizada.

O método set precisa de um parâmetro para funcionar (**Doc**).

Isso apresenta mais segurança ao processo.

```
e = new Escritório
e.setTotalDoc(Doc)
```

Métodos Especiais: Set

Comparando os dois!!

```
e = new Escritório
e.totalDoc = e.totalDoc + 1
```

Veja que o método mais abaixo é mais simplificado e seguro.

```
e = new Escritório
e.setTotalDoc(Doc)
```

vnt/school
powered by venturus

Métodos Especiais

Voltando para o exemplo classe Caneta.

Para simplificar: analise somente os primeiros atributos.

Todos eles apresentam os métodos **get** e **set**.

Caneta
+ modelo + cor - ponta - carga # tampada
+ getModelo() + setModelo(m) + getCor() + setCor(c)

vnt/school
powered by venturus

Métodos Especiais

Todos os **atributos** possuem o método **get** e **set**.

O método **get** deve **retornar** o atributo correspondente.

O método **set** deve receber um **parâmetro**.

```
Classe Caneta
publico modelo: caractere
publico cor: caractere
publico Metodo getModelo()
    retorne modelo
fimMetodo
publico Metodo setModelo(m:caractere)
    modelo = m
fimMetodo
FimClasse
```

vnt/school
powered by venturus

Métodos Especiais

Para construir o objeto.

Pode-se usar os atributos.

Mas é muito mais conveniente e seguro usar os métodos **get** e **set**.

```
c1 = new Caneta
c1.setModelo("Bic")
c1.setCor("Vermelho")
c1.setPonta(0.7)
Escrever(c1.getModelo())
Escrever(c1.getCor())
```

Métodos Especiais

```java
package aula15;
public class Caneta {
    public String modelo, cor;
    private float ponta;
    protected int carga;
    public boolean tampada;

    public String getModelo() {
        return modelo;
    }
    public void setModelo(String modelo) {
        this.modelo = modelo;
    }
    public String getCor() {
        return cor;
    }
    public void setCor(String cor) {
        this.cor = cor;
```

??

Construtor

Construtor é um mecanismo que permite fazer **inicializações** no objeto assim que ele é **declarado**.

São os responsáveis por criar o objeto em memória, ou seja, instanciar a classe que foi definida.

É como iniciar com um **objeto padrão** já estabelecido.

Construtor

Seguindo o exemplo da caneta, pode-se criar uma classe que inicialmente já tem um construtor.

Ao instanciar o objeto obrigatoriamente ele já vai ter o modelo, a cor e a ponta, conforme estabelecido.

Veja o exemplo no NetBeans.

```
Classe Caneta
Metodo construtor(m:caractere, c:caractere, p:real)
    setModelo(m)
    setCor(c)
    serPonta(p)
fimMetodo
FimClasse
```

Construtor

Permite construir o objeto com muito mais praticidade.

```java
package aula15;
public class Caneta {
     public String modelo, cor;
    private float ponta;
    protected int carga;
    public boolean tampada;

    public Caneta(String modelo, String cor, float ponta) {
        this.modelo = modelo;
        this.cor = cor;
        this.ponta = ponta;
        this.tampar();
    }
```

??

vnt/school
powered by venturus

Exercícios

vnt /school
powered by venturus

Atividade 2

Proposta de melhoria:
Ao executar o comando abaixo, instanciamos um "produto" com seus atributos vazios.

product = new Product();

Memória:

nome	preco	quantidade
null	0.0	0

Entretanto, faz sentido um produto que não tem nome? Faz sentido um produto que não tem preço? Com o intuito de evitar a existência de produtos sem nome e sem preço, refaça o programa anterior com a inserção do construtor, tornando obrigatória a iniciação desses valores.

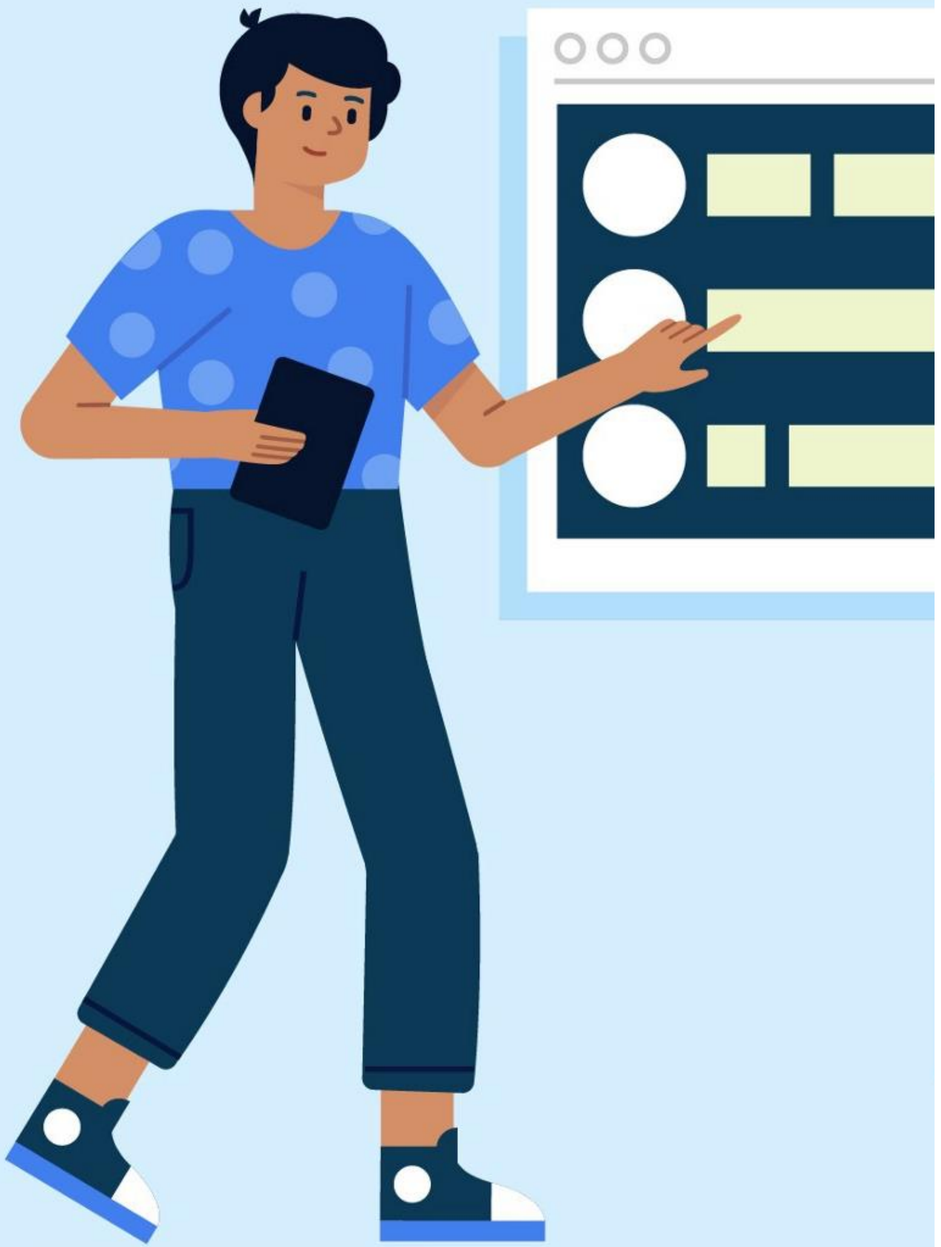
Atividade 2

```java
package aula15;
public class Produto {
    public String nome;
    protected double preco;
    protected int quantidade;

    public Produto(String nome, double preco, int quantidade) {
        this.nome = nome;
        this.preco = preco;
        this.quantidade = quantidade;
    }
```

??


Review
e
Preview



Comunidade VNT

Dica de hoje

O link abaixo apresenta uma breve explicação sobre os conceitos de UML. Este é um tópico muito importante abordado em programação orientada a objeto.

https://spaceprogrammer.com/uml/introducao-as-classes-associacoes-e-generalizacoes/

Boa leitura!!

Referências

[1] A. Goldman, F. Kon, Paulo J. S. Silva; Introdução à Ciência da Computação com Java e Orientação a Objetos (USP). 2006. Ed. USP.
[2] Algoritmo e lógica de programação. Acessado julho/2022: https://visualg3.com.br/
[3] G. Silveira; Algoritmos em Java; Ed. Casa do Código.
[4] M. T. Goodrich, R. Tamassia; Estrutura de dados e algoritmos em Java. Ed Bookman. 2007.
[5] Algoritmo e lógica de programação. Acessado julho/2022: https://www.cursoemvideo.com/
[6] P. Silveira, R. Turini; Java 8 Pratico: lambdas, streams e os novos recursos da linguagem. Ed. Casa do Código.
[7] Linguagem Java: Curso acessado em agosto/2022: https://www.udemy.com/
[8] Linguagem Java: Curso acessado em setembro/2022: https://www.cursoemvideo.com/