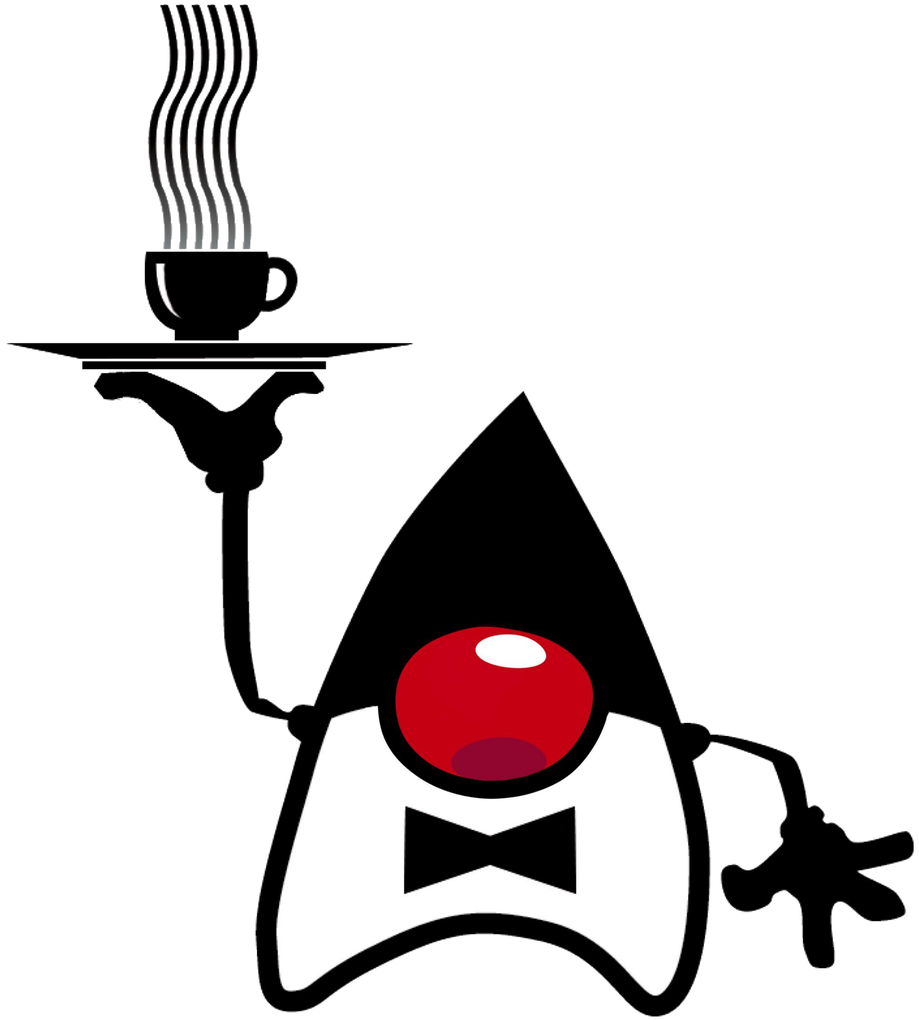
Trilha Java

Encontro 17 – (POO) Encapsulamento



vnt/school
powered by venturus

Recapitulação

1. UML.
2. Visibilidade.

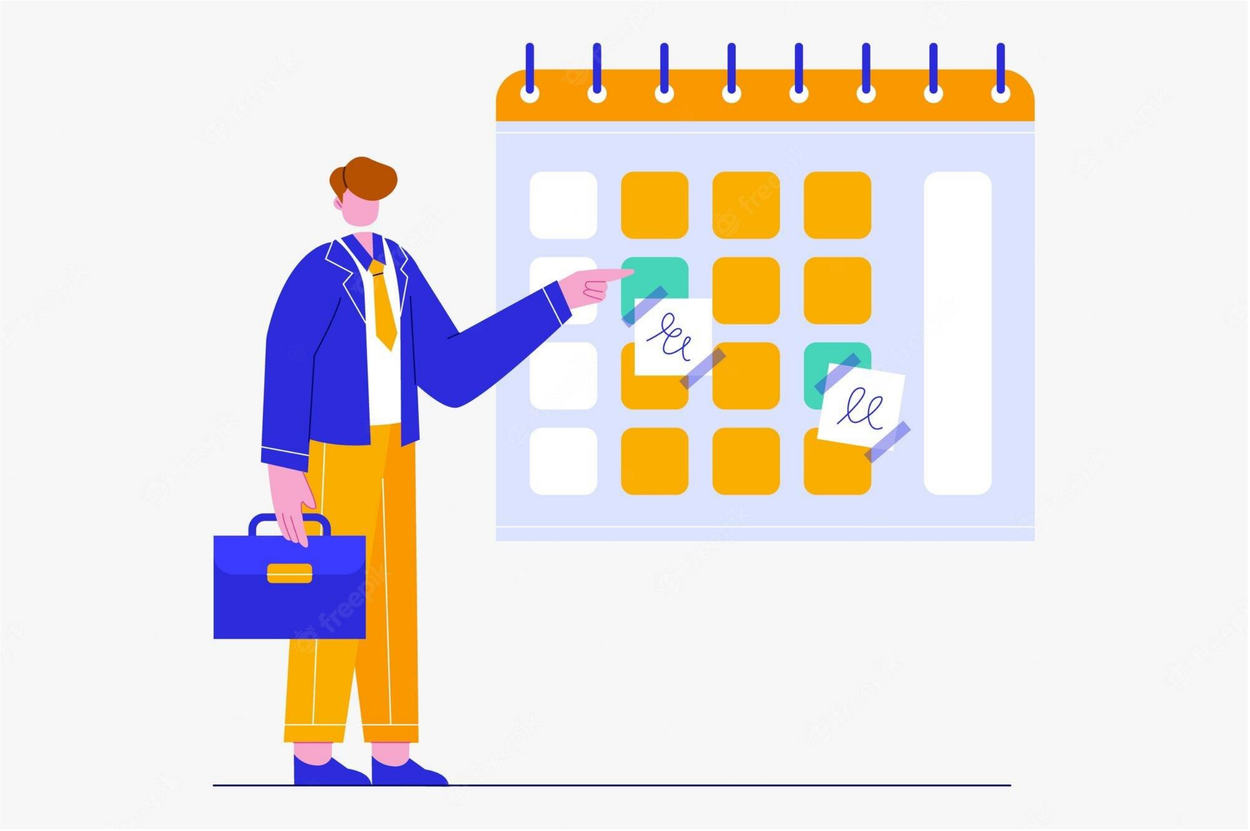
   Atributos e Métodos
3. Métodos Especiais.

   Get, Set, Construtor

vnt/school
powered by venturus

Agenda

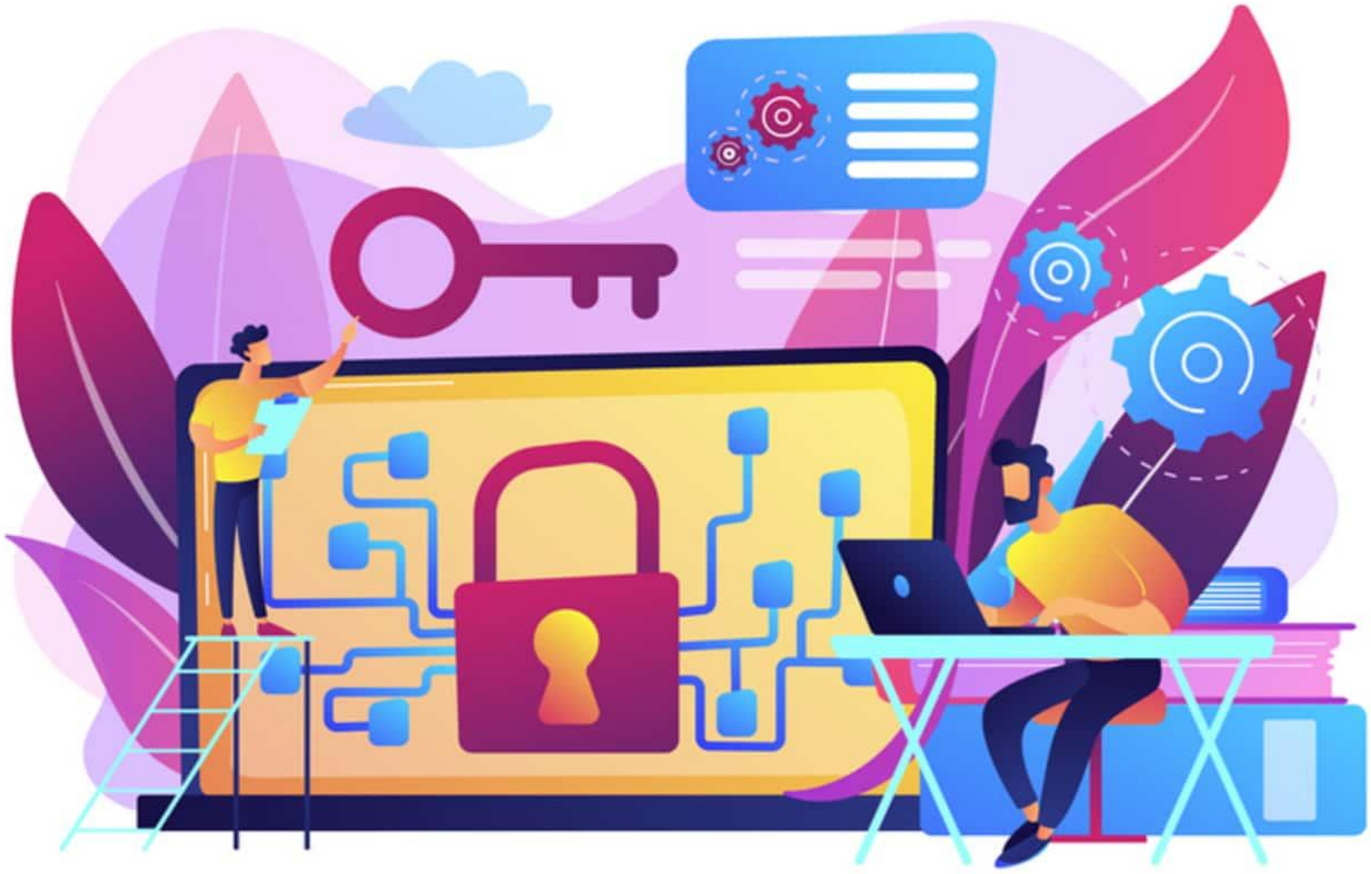
1. Encapsulamento.
2. Relacionamento entre classes.
3. Agregação
4. Exemplos.
5. Atividades.



vnt /school
powered by venturus

Encapsulamento

"Proteção"



vnt/school

powered by venturus

Pilares de POO

O conceito de **abstração** já foi apresentado.

Estes tópicos compõe os **pilares da POO**.

Agora iremos abordar **Encapsulamento!**

vnt/school
powered by venturus

Encapsulamento

A pilha é uma capsula.

Qual o objetivo da capsula?

Proteção contra o componente químico.
Ex: Zn.

Proteção para a pilha de fato funcionar.

vnt/school
powered by venturus

Encapsulamento

Uma forma de criar um **padrão**.

Por exemplo, das pilhas apresentadas ao lado, a **quadrada** **não serve** para o mesmo aparelho que serve as **pilhas palitos**.

Veja que existe um formato, uma intensidade...existe um padrão.

vnt/school
powered by venturus

Encapsulamento

Um software **encapsulado** tem o mesmo **padrão**.

Ele **protege o código** do usuário.

E protege o usuário do código.

vnt/school
powered by venturus

Encapsulamento

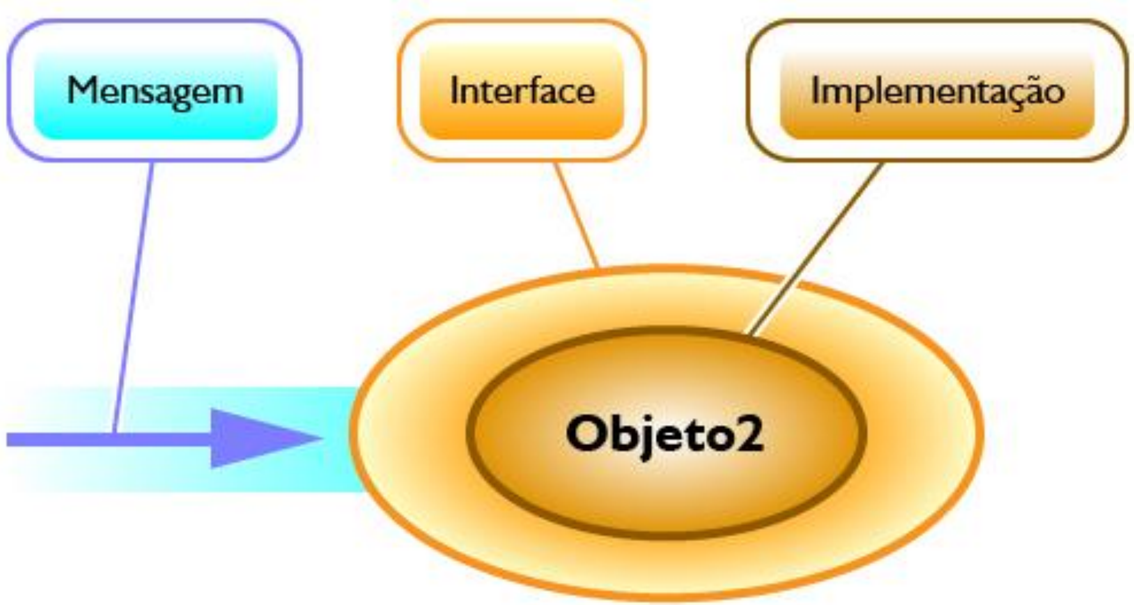
Portanto, **encapsular** é **ocultar** partes independentes da implementação, permitindo construir partes invisíveis ao **mundo exterior**.

vnt/school
powered by venturus

Encapsulamento

Pilha: tem dois pinos que se conecta com o mundo exterior. O **código:** tem contato com o mundo exterior (Interface).

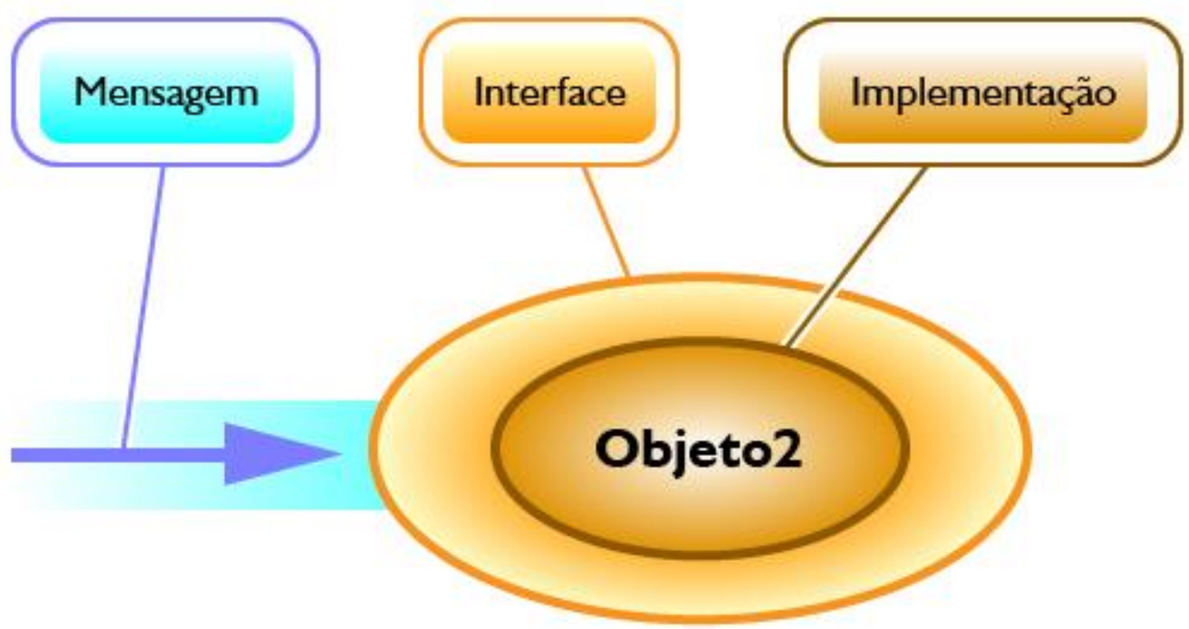
Um **objeto** que possui
a característica de **encapsulamento**
fica protegido por uma cápsula.



Encapsulamento

A cápsula **interface**, serve para ocultar e proteger de outros objetos, os detalhes de implementação daquele objeto.

O **objeto** só disponibiliza, através da sua interface, os serviços que ele deseja oferecer a outros objetos.



Encapsulamento

Importante!!

Encapsular não é obrigatório, mas é uma boa prática para produzir Classes mais eficientes.

Vantagens!!

Tornar mudanças invisíveis.
Facilitar reutilização do código.
Reduzir efeitos colaterais.

Encapsulamento

Uma interface só tem métodos (métodos abstratos)

Os métodos não são desenvolvidos na interface, ou seja, não possui código do seu funcionamento.

Na interface, os métodos são apenas citados.

Todos os métodos na interface são Públicos.

<< interface >>
Controlador

ligar()
desligar()
abrirMenu()
fecharMenu()
maisVolume()
menosVolume()
ligarMudo()
desligarMudo()
play()
pause()

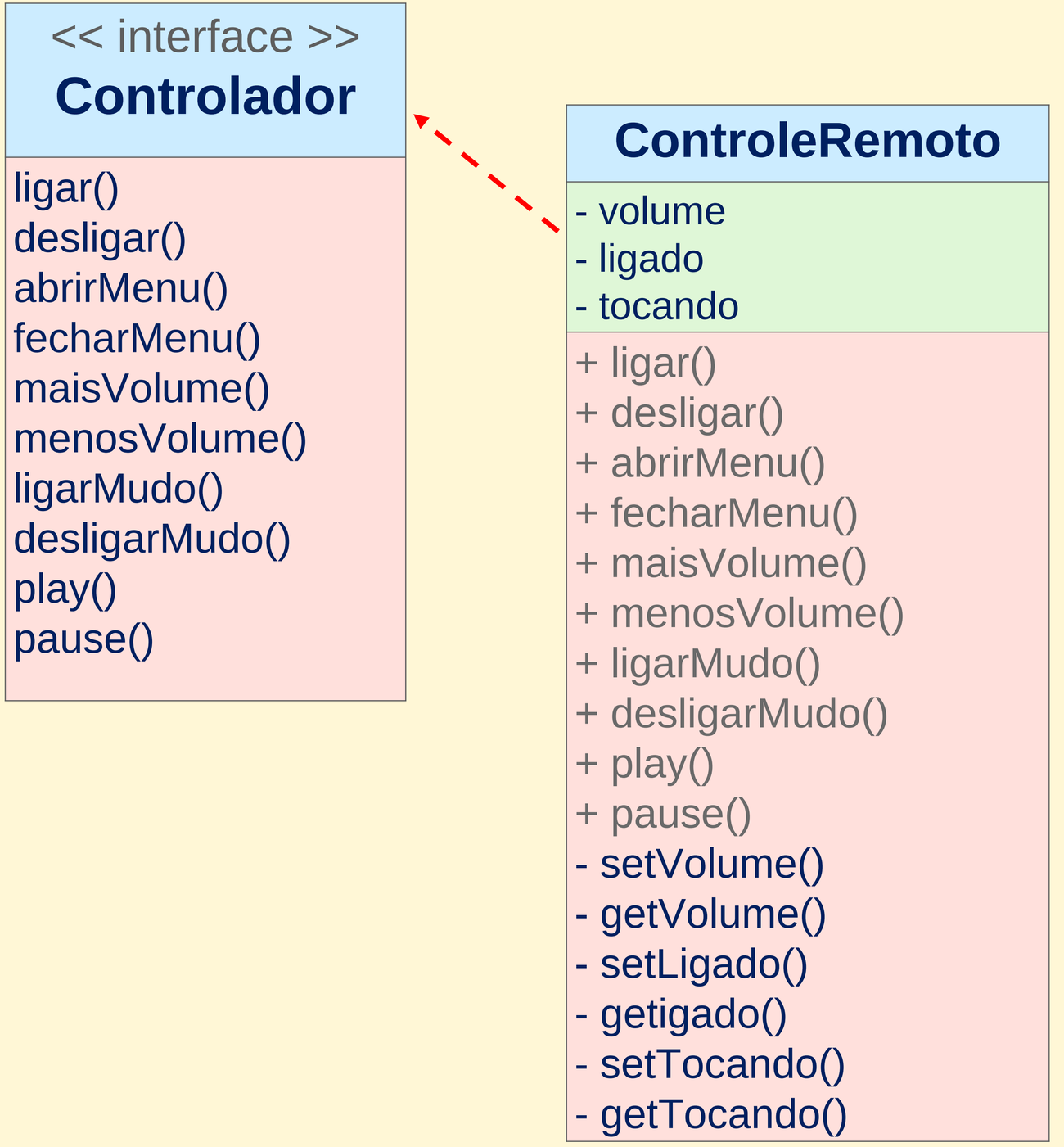
vnt/school
powered by venturus

Encapsulamento

Criamos a Classe “**Controle Remoto**”.

Nesta Classe é implementado o código de cada método do “**Controlador**”.

Neste exemplo vamos deixar os atributos e os métodos Gets e Sets privado.

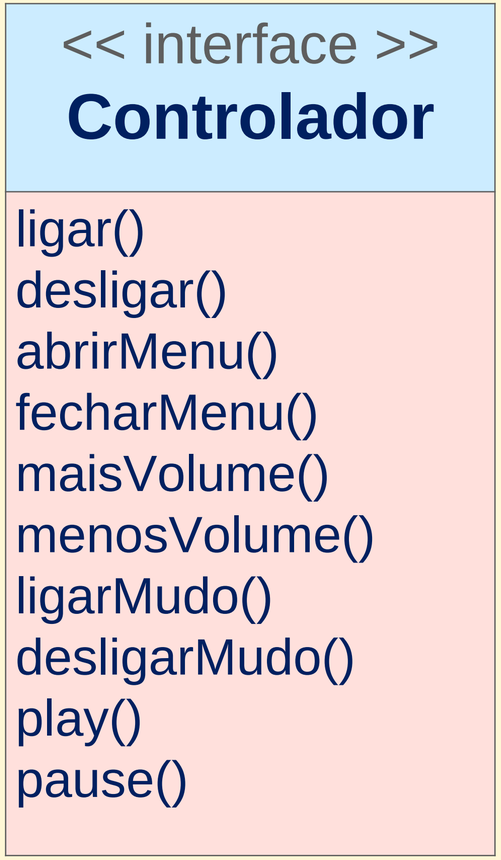


vnt/school
powered by venturus

Encapsulamento

Esta é a **interface**, de acordo com o modelo do **controlador**.

O termo **abstrato** indica que não será implementado o **código na interface**.




```
Interface Controlador
//Métodos Abstratos
publico abstrato Metodo ligar()
publico abstrato Metodo desligar()
publico abstrato Metodo abrirMenu()
publico abstrato Metodo fecharMenu()
publico abstrato Metodo maisVolume()
publico abstrato Metodo menosVolume()
publico abstrato Metodo ligarMudo()
publico abstrato Metodo desligarMudo()
publico abstrato Metodo play()
publico abstrato Metodo pause()
FimInterface
```

vnt/school
powered by venturus

Encapsulamento

Agora basta construir a Classe **ControleRemoto** conforme exemplos anteriores.

O que falta agora?

vnt/school
powered by venturus

ControleRemoto
- volume - ligado - tocando
+ ligar() + desligar() + abrirMenu() + fecharMenu() + maisVolume() + menosVolume() + ligarMudo() + desligarMudo() + play() + pause() - setVolume() - getVolume() - setLigado() - getigado() - setTocando() - getTocando()

```
classe ControleRemoto
//Atributos
privado inteiro volume
privado logico ligado
privado logico tocando
//Metodos Especiais
publico Metodo Construtor()
    volume = 50
    ligado = falso
    tocando = falso
FimMetodo
privado Metodo getVolume()
    retorne volume
fimMetodo
Privado Metodo getLigado()
    retorne volume
fimMetodo
publico  Metodo getTocando()
    retorne volume
FimMetodo
...
FimInterface
```

Encapsulamento

Falta fazer a ligação.

O controle remoto **implementa** o controlador.

E sobrescrever os métodos dentro da classe.

```
classe ControleRemoto
Implementa Controlador
//Atributos
//Metodos Especiais
...
// Sobrescrevendo Metodos
Publico Metodo ligar()
(...)
FimMetodo
Publico Metodo desligar()
(...)
FimMetodo
...
FimInterface
```

vnt/school
powered by venturus

Encapsulamento

Veja no **NetBeans** a implementação do código.

Todos os detalhes são apresentados.

vnt/school
powered by venturus

Encapsulamento

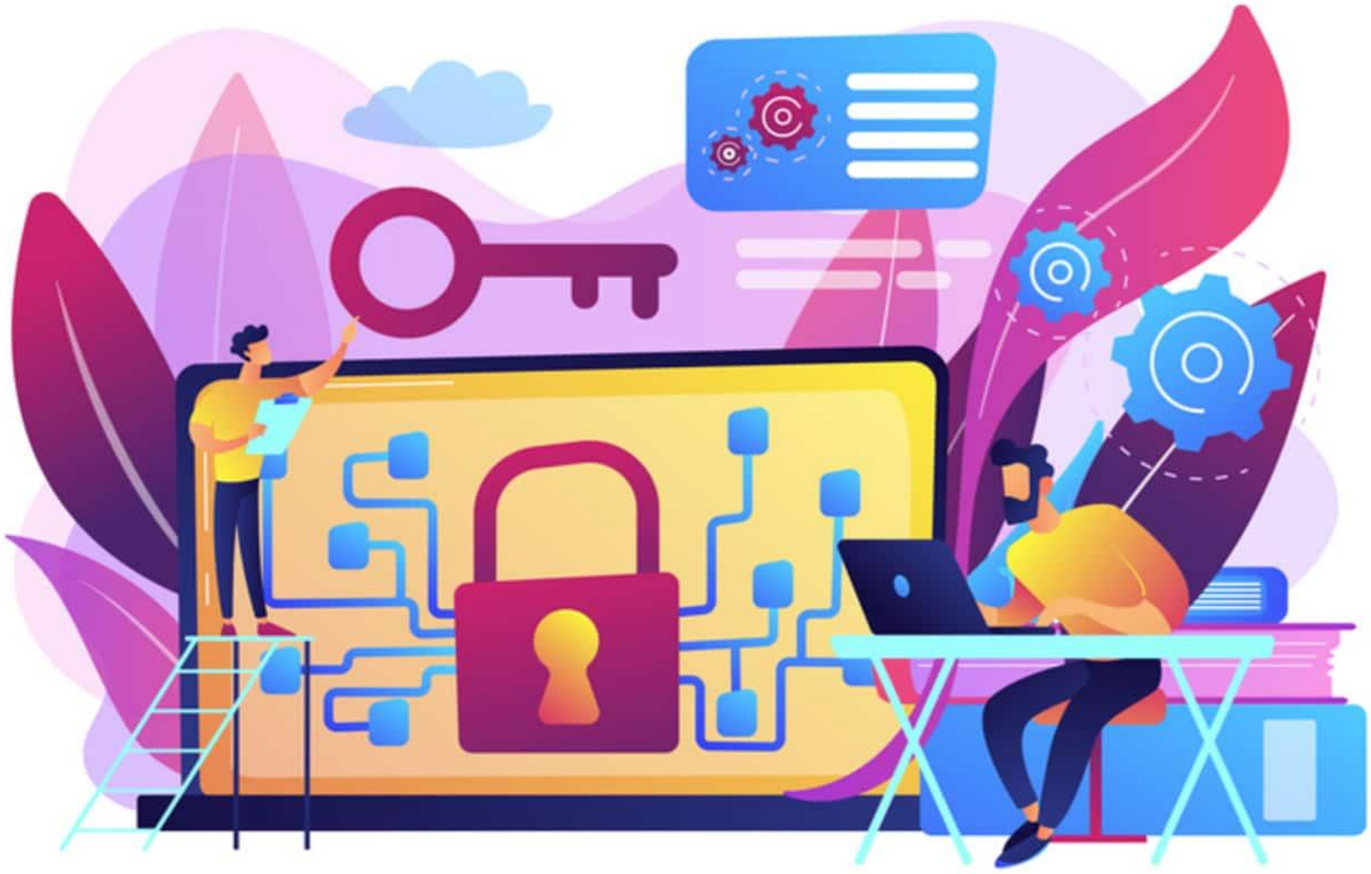
```
package aula17;
public class Exemplo1 {
    public static void main(String[] args) {

        ControleRemoto c = new ControleRemoto();
        c.ligar();
        c.play();
        c.maisVolume();
        c.abrirMenu();
        c.fecharMenu();
    }
}
```

??

vnt/school
powered by venturus

Exercício

Encapsulamento



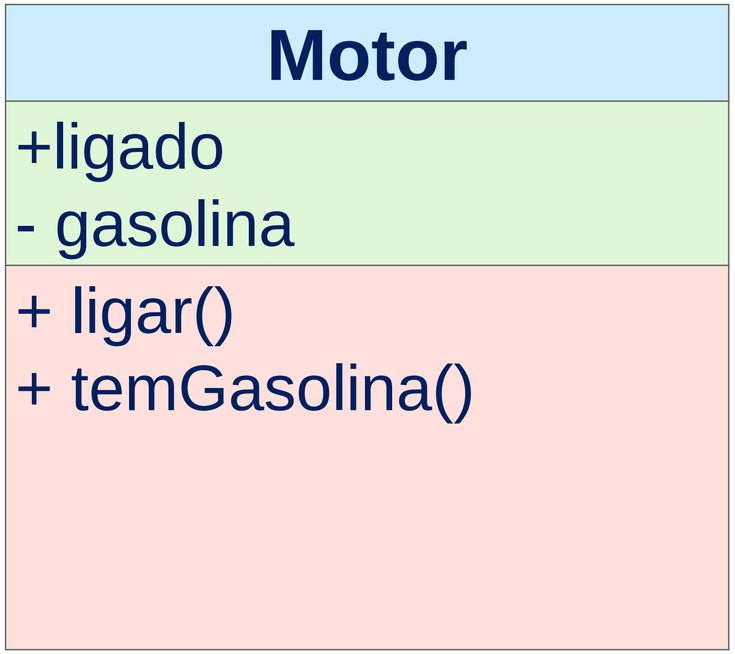
vnt /school

powered by venturus

Exercício 1

Crie um programa que contenha os métodos e atributos conforme diagrama de classe. Imagine que o carro(motor) só deve ligar se este conter gasolina em seu tanque. No entanto, perceba que usar o motor ou até mesmo a gasolina, não deve ser uma ação direta do usuário. Use o conceito de encapsulamento e crie uma interface para os métodos citados. Instancie um objeto motor. Apresente o resultado.

Se tem gasolina, apresente a mensagem "O motor foi ligado", caso contrário indique "não temos gasolina para ligar o motor"



vnt/school
powered by venturus

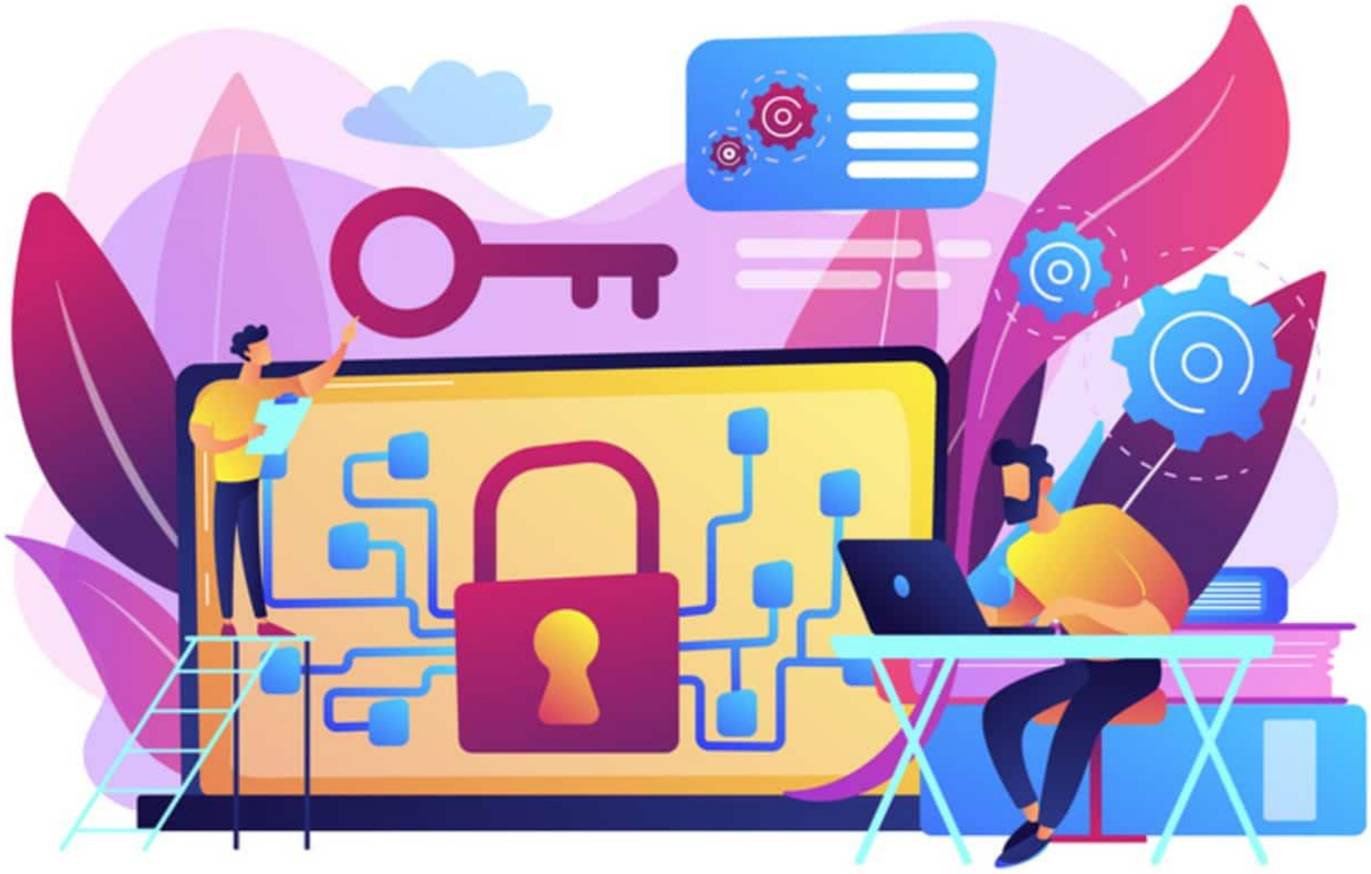
Exercício 1

```
package aula17;
public class Motor implements ControleCarro {
    public boolean Ligado;
    public int gasolina;

    public void Ligar() {
        boolean temGasolina = TemGasolina();
        if (temGasolina) //a mesma coisa que if(temGasolina == true)
        {
            this.Ligado = true;
            System.out.println("O motor foi ligado.");
        } else {
```

??

coffee
time

Relacionamento Entre Classes



vnt /school
powered by venturus

Relacionamento entre Classes

Podendo as **Classes** serem consideradas engrenagens ou peças de um quebra-cabeça, uma questão surge naturalmente:

Como essas peças poderiam ser relacionadas?

vnt/school
powered by venturus

Relacionamento entre Classes

Se desejarmos **desmontar** um **automóvel**, por exemplo, verificaremos que ele é formado por **peças** que apresentam relações muito definidas.

Esses **elementos**, associados ou conectados de maneira correta, permitem a existência e o correto **funcionamento do automóvel**.

vnt/school
powered by venturus

Relacionamento entre Classes

Suponha uma **luta** entre **lutadores**.

Para tal é necessário construir um **perfil** de cada lutador.

Nome, nacionalidade, idade, altura, peso, categoria, vitórias, derrotas e empates.

vnt/school
powered by venturus

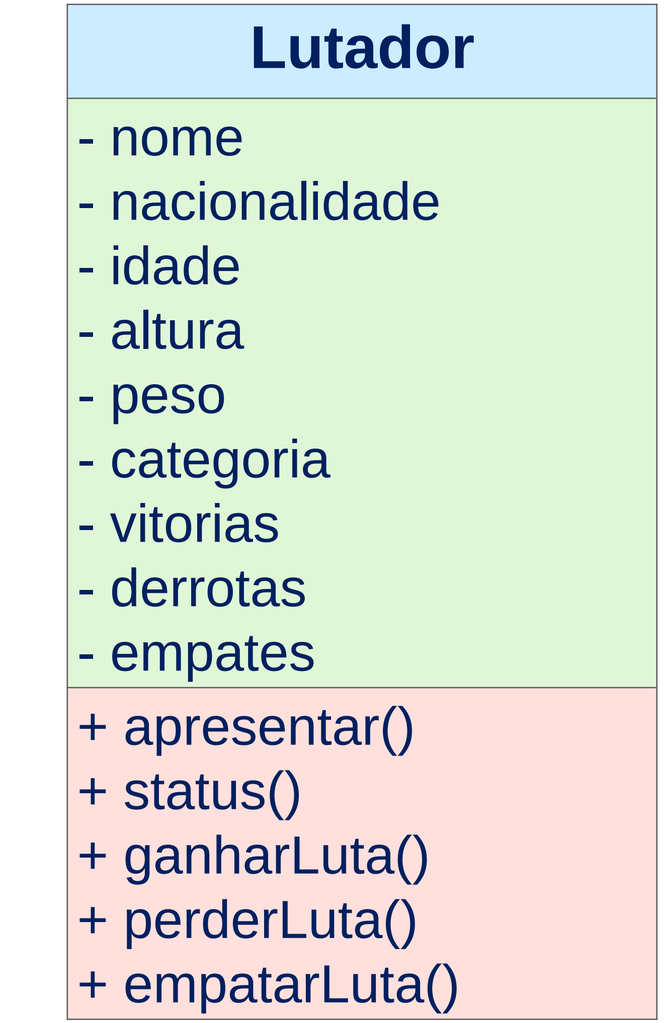
Relacionamento entre Classes

Criamos um diagrama de classe.

Inserimos os **atributos** e **métodos**.

Atribuímos a **visibilidade**.

E ainda podemos inserir os métodos **gets** e **sets**.



Relacionamento entre Classes

Inserir o **construtor** será importante neste projeto.

Neste caso, a **categoria** vai ser inserida de forma automática pelo código.

O código será dentro do **setPeso**.

```
Publico metodo construtor(
no, na, id, al, pe, vi, de, em)
    Nome = no
    Nacionalidade = na
    Idade = id
    Altura = al
    SetPeso(pe)
    Vitorias = vi
    Derrotas = de
    Empates = em
FimMetodo
```

Relacionamento entre Classes

Além dos
métodos: apresentar(), status(), ga
nharLuta(), perderLuta()
e empatarLuta().

Em especial temos o
método **setCategoria,** que atribui a
definição dos tipos de lutadores.

```
Publico metodo setCategoria( )
    Se(peso < 52.2)entao
        Categoria = "Inválido"
    Senao se(peso <= 70.3)entao
        Categoria = "Leve"
    Senao se(peso <= 83.9)entao
        Categoria = "Medio"
    Senao se(peso <= 120.2)entao
        Categoria = "Pesado"
    Senao
        Categoria = "Invalido"
    fimSe
FimMetodo
```

vnt/school
powered by venturus

Relacionamento entre Classes

Como instanciar o objeto?

Podemos criar um vetor!

```
//Programa Principal
L: vetor[ 0..3]
L[0] = new Lutador(no, na, id, al, pe, vi, de, em)
L[1] = new Lutador(no, na, id, al, pe, vi, de, em)
L[2] = new Lutador(no, na, id, al, pe, vi, de, em)
L[3] = new Lutador(no, na, id, al, pe, vi, de, em)
```

Após instanciar o obejto, você pode "chamar" cada objeto conforme desejado.

```
//Programa Principal
L: vetor[ 0..3]
L[0].apresentar( )
L[1].status( )
L[2].getCategoria( )
```

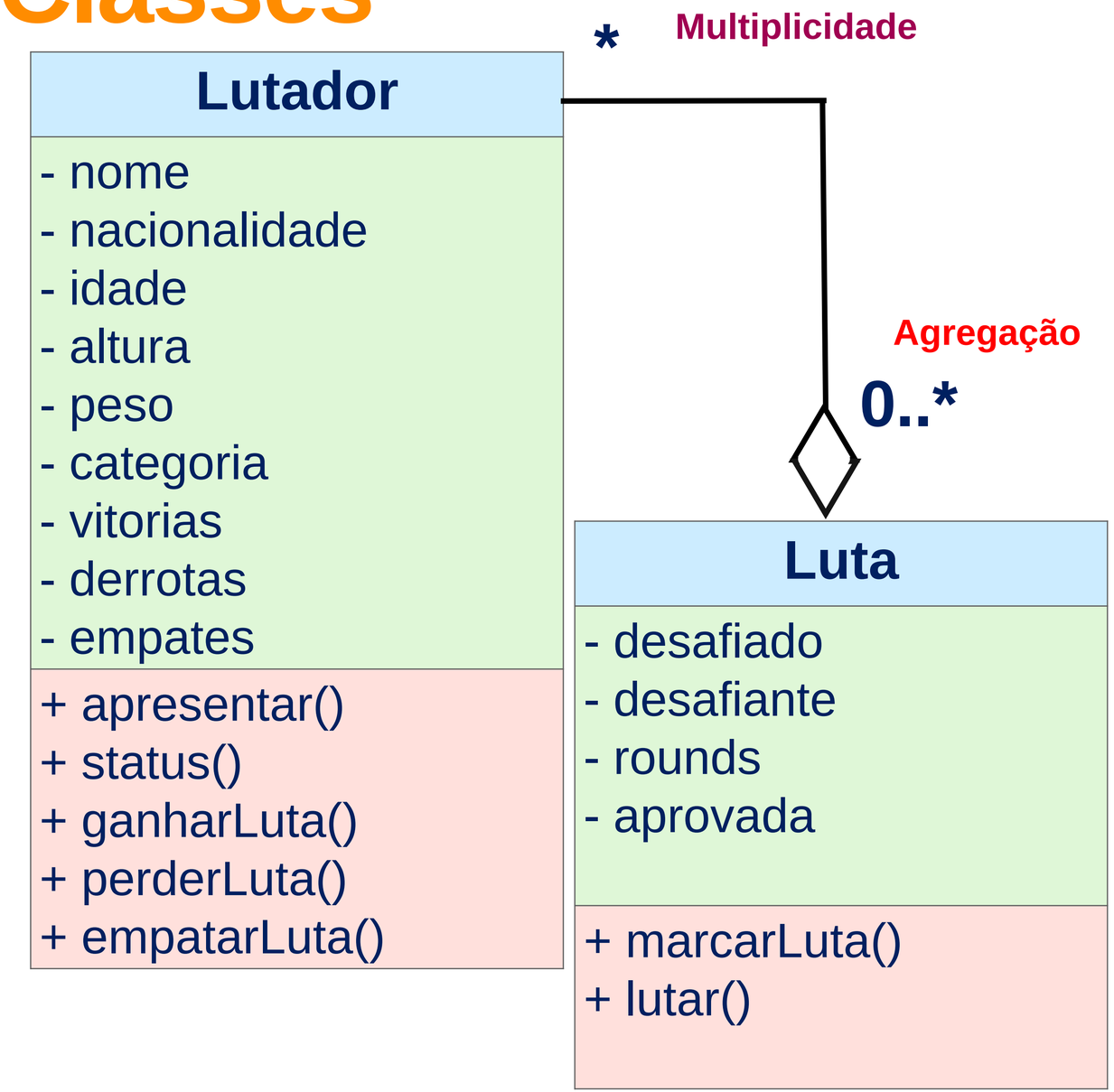
Relacionamento entre Classes

Como funciona o relacionamento de classes?

Uma nova classe “Luta” pode ser criada. Pois, todo lutador pode participar de uma luta.

Toda vez que você identificar um objeto que é parte de outro objeto, você tem uma agregação.

Um objeto pode agregar um ou muitos objetos. Isso se chama multiplicidade.



vnt/school
powered by venturus

Relacionamento entre Classes

Regras de uma luta!

Só pode ser marcado entre lutadores de uma mesma categoria.

Desafiado e desafiante devem ser lutadores diferentes.

Só pode acontecer se estiver aprovada.

Só pode ter como resultado a vitória de um dos lutadores ou empate.

vnt/school
powered by venturus

Relacionamento entre Classes

Veja que na classe **Luta**,
o **atributo** **desafiado** e **desafiante**
é do tipo “**Lutador**”

É possível usar os **gets**
de outra **classe**, pra **marcarLuta**.

vnt/school
powered by venturus

Classe Luta
//Atributos
Privado desafiado: Lutador
Privado desafiante: Lutador
Privado rounds: inteiro
Privado aprovada: Lógico
...
Public Metodo marcarLuta(l1, l2 : Lutador**)**
Se ((l1.getCategoria() = l2.getCategoria() **)**
e(l1 <> l2**)) Então**
Aprovada = true
desafiado = l1
desafiante = l2
Senão
Aprovada = false
desafiado = nulo
desafiante = nulo

Relacionamento entre Classes

Desenvolvendo o método “lutar”.

Aqui é feito as definições de empate, vitória e derrota.

```
Classe Luta
//Metodos
Public Metodo Lutar( )
Se (Aprovada)
    Desafiado.apresentar()
    Desafiante.apresentar()
    vencedor = aleatório(0..2)
    Escolha (Vencedor)
        Caso 0: Empate
        Caso 1: desafiado ganha
        Caso 2: desafiante ganha
    fimEscolha
Senão
"Luta não pode acontecer"
fimSe
```

Relacionamento entre Classes

Por fim, é incrementado o programa final.

E instanciado o objeto luta.

```
//Programa Principal
L: vetor[ 0..3]
L[0] = new Lutador(....)
L[1] = new Lutador(....)
L[2] = new Lutador(....)
L[3] = new Lutador(....)

UFC01 = new Luta( )
UFC01.marcarLuta(L[0], L[1])
UFC01.lutar( )
```

Relacionamento entre Classes

Veja no **NetBeans** a implementação do código.

Todos os detalhes são apresentados.

vnt/school
powered by venturus

Relacionamento entre Classes

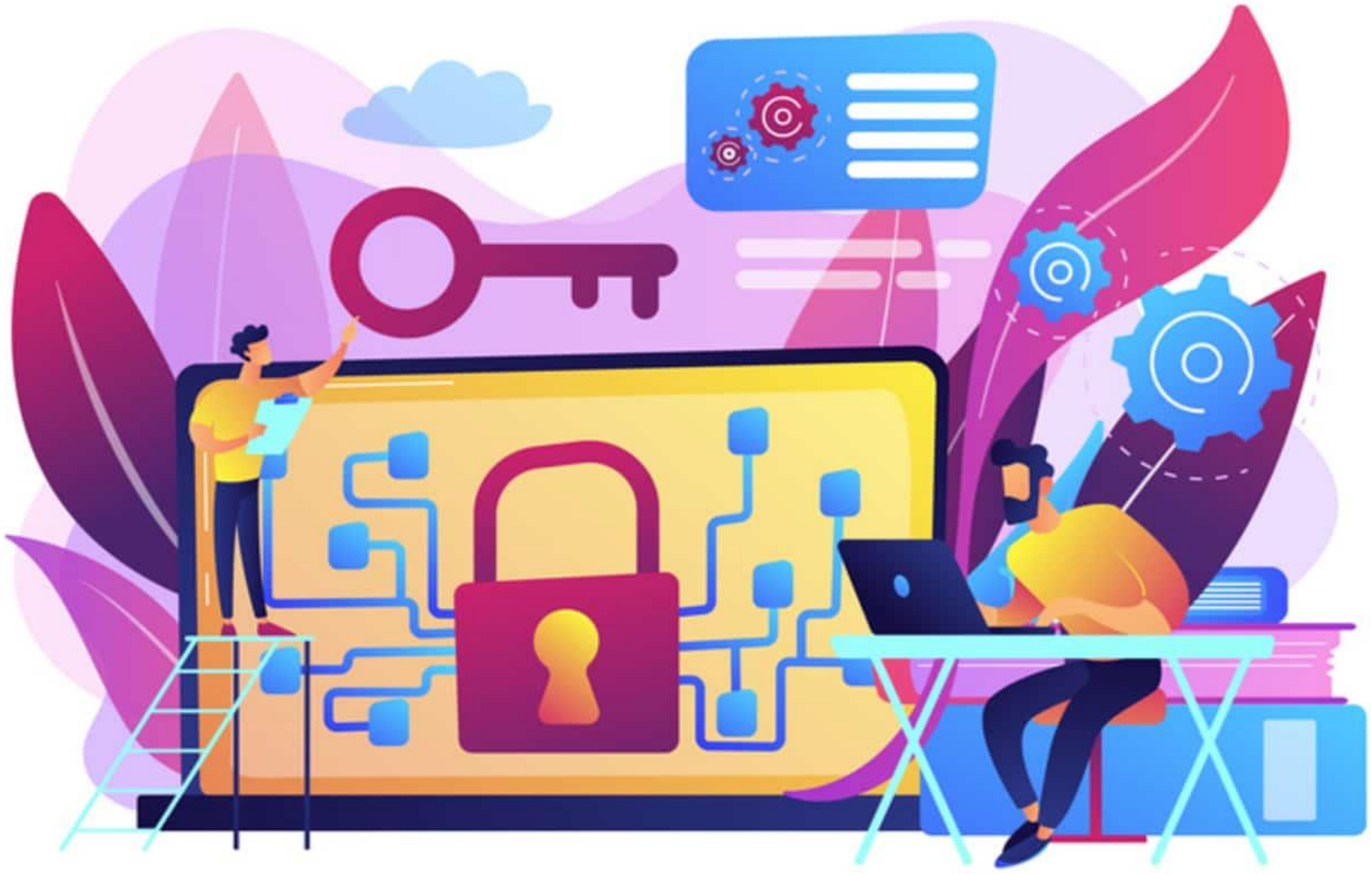
```
package aula17exemplo2;
public class Aula17Exemplo2 {
    public static void main(String[] args) {
        Lutador l[] = new Lutador[6];
        l[0] = new Lutador("BOB", "França", 1.75f, 68.9f, 31, 11, 2, 1);
        l[1] = new Lutador("CADU", "Brasil", 1.68f, 57.8f, 29, 14, 2, 3);
        l[2] = new Lutador("STEPHEN", "EUA", 1.65f, 80.9f, 35, 12, 2, 1);
        l[3] = new Lutador("RICHARD", "Austrália", 1.93f, 81.6f, 28, 13, 0, 2);
        l[4] = new Lutador("PEDRO", "Brasil", 1.70f, 119.3f, 37, 5, 4, 3);
        l[5] = new Lutador("PITTER", "EUA", 1.81f, 105.7f, 30, 12, 2, 4);

        Luta UFC = new Luta();
        UFC.marcarLuta(l[4], l[5]);
        UFC.lutar();
    }
}
```

??

vnt/school
powered by venturus

Execício

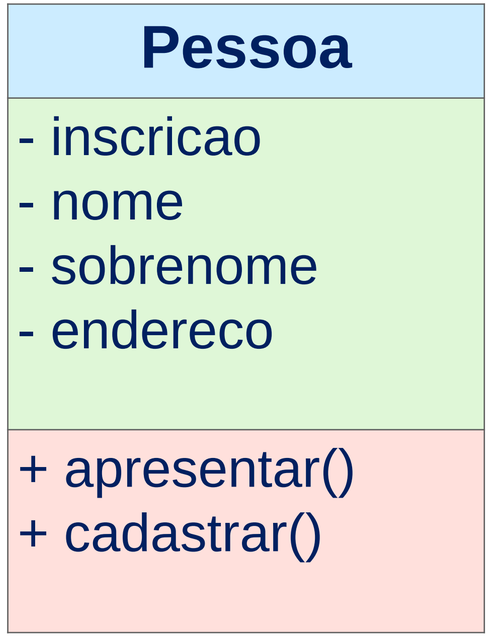
Relacionamento entre Classes

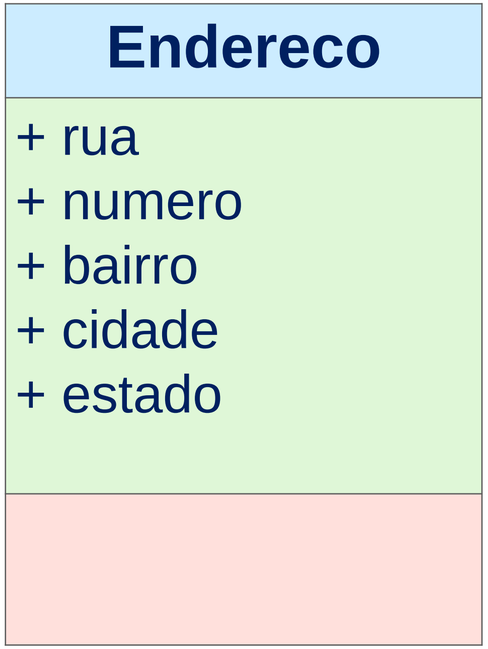


vnt/school
powered by venturus

Exercício 2

Faça um programa com as características apresentadas no diagrama de classe e acrescente o que achar necessário. Apresente o resultado no final criando um objeto "**pessoa**". Use a ideia do relacionamento entre Classes. Suponha que num futuro próximo essa empresa pretende usar diferentes classes para pessoas(funcionarios, clientes, fornecedores e etc), e todas essas pessoas irão se beneficiar da classe "**Endereco**". Portanto, ter uma classe separada que defina somente ela, poderá ser usada mais facilmente.



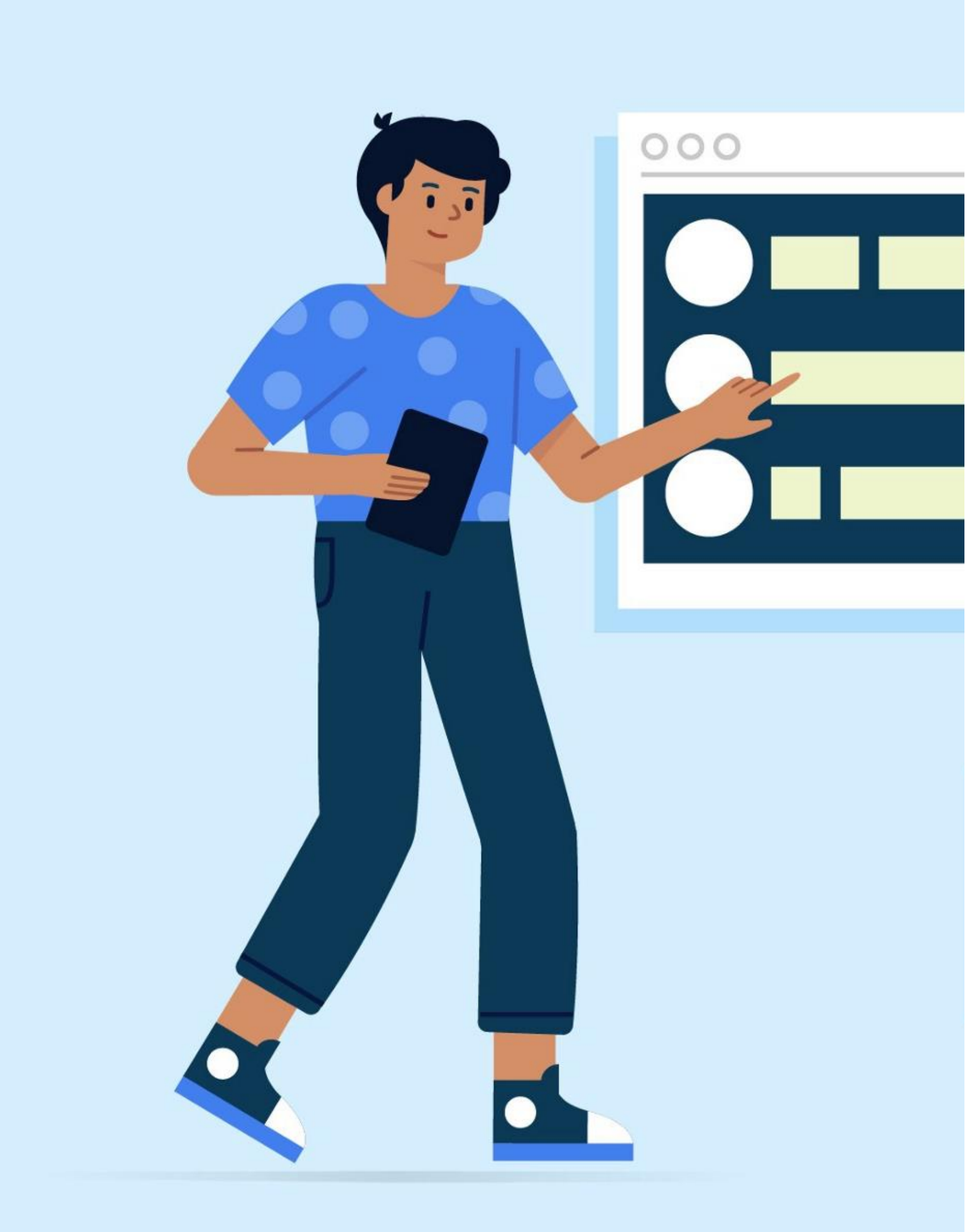


vnt/school
powered by venturus

Exercício 2

```java
package aula17;
public class Exercicio2 {
    public static void main(String[] args) {
        Pessoa p1 = new Pessoa();
        p1.cadastrar();
        p1.end = new Endereco();
        p1.end.rua = "Avenida Paulista";
        p1.end.numero = 14;
        p1.end.bairro = "Bela Vista";
        p1.end.cidade = "Sao Paulo";
        p1.end.estado = "SP";
        p1.apresentar();
    }
}
```


Review
e
Preview



Comunidade VNT

Dica de hoje

O link abaixo é um dos sites mais importantes da comunidade de tecnologia.
Ele tem como objetivo ajudar na solução de problemas em diferentes níveis.
É uma plataforma ideal para levantar dúvidas e debates.
Surgiu dúvidas na solução de problemas com a linguagem Java ou outra qualquer, procure ajuda no **Stackoverflow**!!

https://stackoverflow.com/

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/