

CURSO JAVA – CURSO EM VIDEO

[Programacão Orientada a Objetos 4](#_Toc17722727)

[Requisitos: 4](#_Toc17722728)

[História Java: 4](#_Toc17722729)

[Objetivo: 4](#_Toc17722730)

[Quem criou: 4](#_Toc17722731)

[Dynabook: 4](#_Toc17722732)

[Vantagens: (COMERN) 5](#_Toc17722733)

[O que é um objeto? 7](#_Toc17722734)

[Caneta: 7](#_Toc17722735)

[Classe: 7](#_Toc17722736)

[Objeto: 8](#_Toc17722737)

[Abstração: 9](#_Toc17722738)

[Mudando visibilidade 11](#_Toc17722739)

[Linguagem de Modelagem Unificada (UML): 11](#_Toc17722740)

[Diagrama de Classe: 11](#_Toc17722741)

[Modificadores de Visibilidade: 11](#_Toc17722742)

[Público 11](#_Toc17722743)

[Privado 11](#_Toc17722744)

[Protegido 11](#_Toc17722745)

[Padrão 11](#_Toc17722746)

[métodos especiais 13](#_Toc17722747)

[Métodos Acessores (getters) 13](#_Toc17722748)

[Métodos Modificadores ou mutantes (Setters) 13](#_Toc17722749)

[Métodos Construtores: 16](#_Toc17722750)

[Pilares da POo 19](#_Toc17722751)

[Encapsulamento 19](#_Toc17722752)

[O que é encapsular? 19](#_Toc17722753)

[Interface 19](#_Toc17722754)

[Aviso 19](#_Toc17722755)

[Vantagem 19](#_Toc17722756)

[Como encapsular? 19](#_Toc17722757)

[Para criar em interface: 20](file:///C:\Users\Felipe%20Avelar\Studies\Java\Java%20-%20Curso%20em%20vido.docx#_Toc17722758)

[Métodos Abstratos 20](file:///C:\Users\Felipe%20Avelar\Studies\Java\Java%20-%20Curso%20em%20vido.docx#_Toc17722759)

[O que acontece quando se encapsula 20](file:///C:\Users\Felipe%20Avelar\Studies\Java\Java%20-%20Curso%20em%20vido.docx#_Toc17722760)

[Utilização 20](file:///C:\Users\Felipe%20Avelar\Studies\Java\Java%20-%20Curso%20em%20vido.docx#_Toc17722761)

[Relacionamento entre classes 27](#_Toc17722762)

[Agregação 27](#_Toc17722763)

[Papel 27](file:///C:\Users\Felipe%20Avelar\Studies\Java\Java%20-%20Curso%20em%20vido.docx#_Toc17722764)

[Sentido 27](file:///C:\Users\Felipe%20Avelar\Studies\Java\Java%20-%20Curso%20em%20vido.docx#_Toc17722765)

[Multiplicidade 27](file:///C:\Users\Felipe%20Avelar\Studies\Java\Java%20-%20Curso%20em%20vido.docx#_Toc17722766)

Programacão Orientada a Objetos

Requisitos:

* Algoritmo;
* Java básico;

História Java:

POO – Programa Orientada a Objetos (OOP inglês).

Objetivo: Aproximar o mundo digital do mundo real.

Como era antes da POO:

1. 1940: Programação de baixo nível (mudava de cada máquina);
2. Programação Linear (Começo do alto nível);
3. Programação Estruturada (Permitia pequenos passos da programação linear fossem executados fora de sua ordem natural; Início dos sistemas);
4. Programação Modular (Permitia que criasse pequenos módulos estruturados, valorizando dados e funcionalidades, e colocar eles em pequenas capsulas protegidas, que poderia compor sistemas maiores – Teve vida curta);
5. Orientada a Objetos;

Quem criou: Alan Kay, formado em matemática e biologia, focado na educação infantil, a maioria de seus projetos foram focados na educação. A sua formação em matemática lhe deu um raciocínio logico muito bom, e sua formação em biologia deu origem a grande maioria dos conceitos de POO, criado por ele. Isso ocorreu me 1970.

Dynabook: É um protótipo criado pelo Alan Key, que é um dos conceitos iniciais do laptop. Porém não havia tecnologia suficiente na época, e com isso se criou novas tecnologias, como uma linguagem de programação. Por que segundo ele mesmo(Alan Kay) as linguagens da época não teria suporte com os conceitos do Dynabook, as coisas no Dynabook teriam que ser dinâmicas, tradada como objetos; pensando nisso Alan Kay comandou sua equipe na criação da primeira POO, a Smalltalk, que já contava com conceitos de classes, objetos, atributos, métodos.



Vantagens: (COMERN)

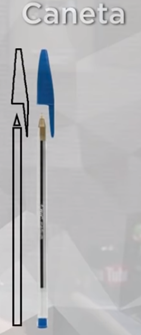
1. Confiável – O isolamento entre as partes gera software seguro. Ao alterar uma parte, nenhuma outra é alterada.
2. Oportuno – Ao dividir tudo em partes, varias delas podem ser desenvolvidas em paralelo.
3. Manutenível – Atualizar um software é mais fácil. Uma pequena modificação vai beneficiar todas as partes que usarem o objeto.
4. Extensível – O Software não é estático. Ele deve crescer para permanecer útil.
5. Reutilizável – Podemos usar objetos de um sistema que criamos, em outro sistema no futuro.
6. Natural – Mais fácil de entender. Você se preocupa mais na funcionalidade do que nos detalhes de implementação.

O que é um objeto?

Coisa material ou abstrata que pode ser percebida pelos sentidos e descrita por meios das suas características, comportamentos e estado atual.

Exemplos: Compromisso, carro, pessoa, caneta.

Caneta:

Todos esses objetos estão baseados na mesma classificação (Como se fosse uma forma). A partir disso, temos duas coisas diferentes: Sendo elas o modelo da caneta, e a caneta em si.  

Para POO a caneta em si é um OBJETO, já esse molde que serve para classificar, como o nome já diz, é uma classe.

Classe:

Sempre tem que responder três perguntas:

1. Coisas que eu tenho? - Atributos

Modelo, Cor, Ponta, Carga, Tampada.

1. Coisas que eu faço? – Métodos

Escrever, Rabiscar, Pintar, Tampar, Destampar.

1. Como estou agora? - Estado

50% de carga, ponta fina, azul, destampada, escrevendo.

Define os atributos e métodos comuns que serão compartilhados por um objeto.

Exemplo de classe:

Classe Caneta

//Atributos

Modelo: Caractere;

Cor: Caractere:

Ponta: Real;

Carga: Inteiro;

Tampado: Logico;

//Metodos

Metodo Rabicar ():

Se (Tampada):

Escreva (“Erro”);

Senao:

Escreva(“Rabisco”);

FimMetodo;

Metodo Tampar ():

Tampada = Verdadeiro;

FimMetodo;

FimClasse;

Objeto:

É a instancia de uma classe.

Exemplo de Objeto:

Instanciando um objeto (Criando um novo):

C1 = nova Caneta;

C1.Cor = “Azul”;

C1.Ponta = 0.5;

C1.Tampada = Falso;

C1.Rabiscar();

C2 = nova Caneta;

C2.Cor = “Vermelho”;

C2.Ponta = 1.0;

C2.Tampada = Falso;

C2.Tampar();

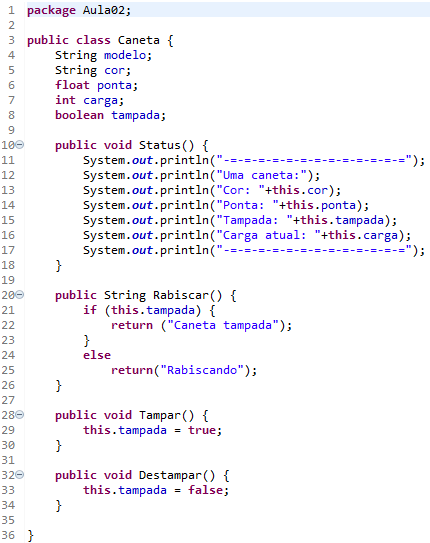


Abstração:

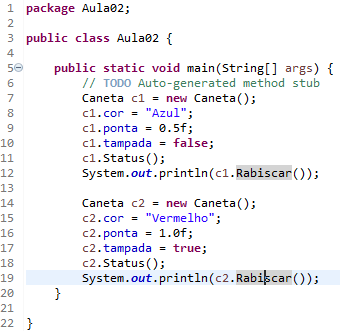
É o ato de abstrair atributos, métodos, apenas que forem necessários para o Software. Deixando de lado o que não for importante.

“O ser Humano é uma coisa muito completa, tem muitos atributos, quais são os atributos que me importam nesse momento?”

Modelo de classe no Java:



Modelo de instanciar um objeto, e utilizar na classe Main.



Mudando visibilidade

Linguagem de Modelagem Unificada (UML):

Muito importante em POO;

Diagrama de Classe:

**Caneta**

Atributos

**Classe**

+ modelo

+ cor

- ponta

# carga

- tampada

Métodos ()

+ status ()

+ escrever ()

+ rabiscar ()

- pintar ()

+ tampar ()

+ destampar ()

Modificadores de Visibilidade:

Indicam o nível de acesso dos componentes internos de uma classe (Atributos e métodos).

+: público/public

-: privado/private

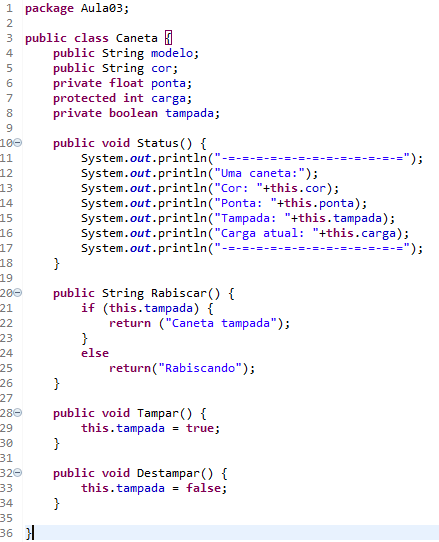
#: protegido/protected

Público: A classe atual e todas as outras classes podem ter acesso.

Privado: Somente a classe atual tem acesso.

Protegido: A classe atual, e todas as suas subclasses.

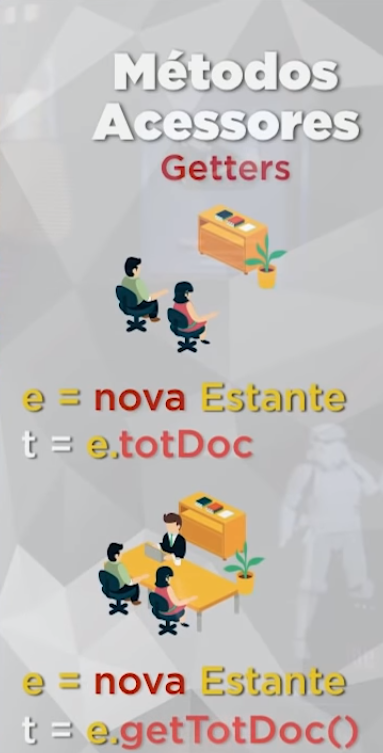
Padrão: Toda linguagem tem sua definição padrão. No caso do Java é pública para pacote.



métodos especiais

Métodos Acessores (getters)

São métodos que conseguem acessar um determinado atributo, mantendo a segurança de acesso a ele.



Imagine que temos um cofre de banco.

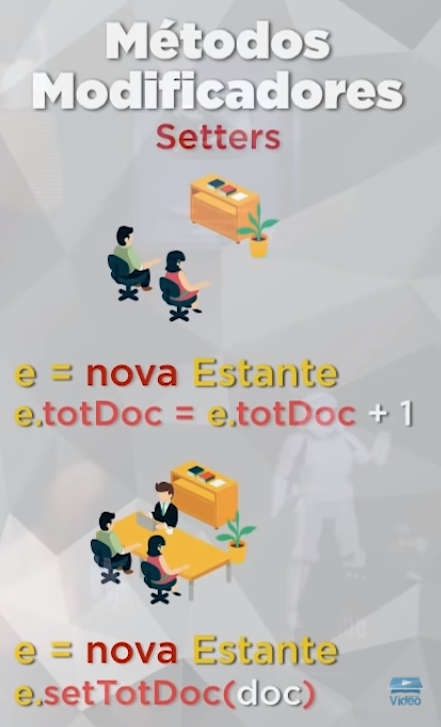
Não podemos deixar acesso livre a todos dentro do cofre, isso seria extremamente arriscado.

Então temos uma pessoa que tem o acesso, e que administra isso em segurança para nós.

Assim garantindo o seu acesso, e a segurança de seu cofre.

Métodos Modificadores ou mutantes (Setters)

São métodos que conseguem alterar um, ou mais determinados atributos, mantendo a segurança dos mesmos.

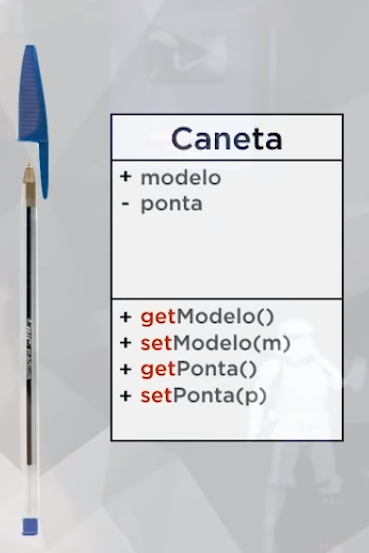


Seguindo o exemplo do cofre.

Imagine que queremos depositar algo nele.

O banco não pode deixar que a gente chegue lá, e simplesmente, nós mesmo abrimos os cofres, e coloquemos nosso deposito dentro.

Para garantir a segurança temos uma pessoa, que tem o acesso, a qual passamos os nossos dados e o valor que queremos depositar, e ele se responsabiliza por depositar nosso dinheiro.



Bic Cristal

0.5

Classe Main

C1 = nova Caneta;

C1.setModelo (“BIC Cristal”);

C1.setPonta (0.5);

C1.getModelo();

C1.getPonta();

Escreva(C1.getModelo());

Escreva(C1.getModelo());

Classe Caneta

//Atributos

public modelo: Caractere;

private ponta: Real;

//Metodos

public metodo getModelo():

retorne this.modelo

fim metodo;

public metodo setModelo( m : Caractere):

this.modelo = m;

fim metodo;

public metodo getPonta():

retorne this.ponta

fim metodo;

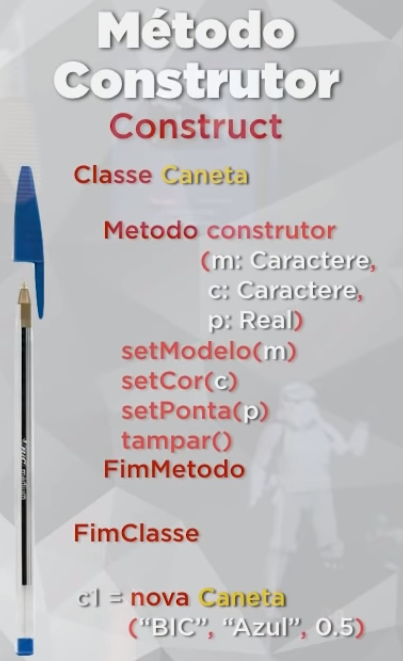
public metodo setPonta( p : Real):

this.ponta = p;

fim metodo;

Métodos Construtores:

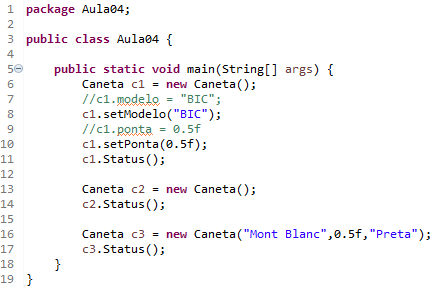
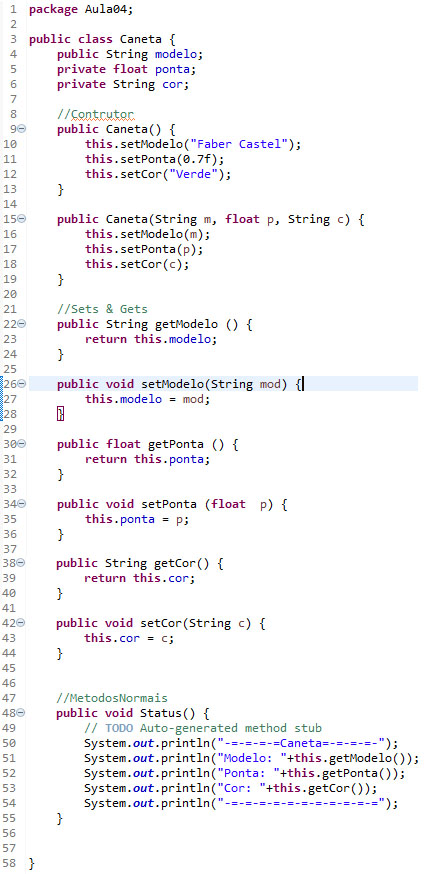
São métodos que definem que ações devem ser executadas na criação de um objeto.



Em Java o construtor é definindo como um método cujo o nome é o mesmo da classe e sem identificação de retorno (nem mesmo o void). O construtor é unicamente invocado no momento da criação do objeto através do operador new.

<http://www.dca.fee.unicamp.br/cursos/PooJava/metodos/construtor.html>

Exemplo de contrutor, e metodos especiais a seguir: o fato de ter dois contrutores, ainda é algo a ser ensinado, apenas levem em base como é feito um construtor.



Resultado:



Pilares da POo

* A – Abstração\*.
* E – Encapsulamento.
* H – Herança.
* P – Polimorfismo.

Algumas bibliografias a usam como um pilar, outras incluem ela dentro do encapsulamento.

Encapsulamento

O que é encapsular?

Ocultar partes independentes da implementação, permitindo construir partes invisíveis ao mundo exterior.

Considere como uma cápsula. Exemplo uma pilha.

Por que pilha é uma capsula?

* Para proteger você dos componentes químicos que estão dentro.
* E uma proteção da pilha de você, para que não altere algo que a estrague.
* Criar um padrão.

Interface

Lista de serviços fornecidos por um componente. É o contato com o mundo exterior, que define o que pode ser feito com um objeto dessa classe.

Exemplo: polos da pilha.

Aviso

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

Vantagem

* Tornar mudanças invisíveis.
* Facilitar a reutilização de código.
* Reduzir efeitos colaterais.

Como encapsular?

Para criar em interface:

Não possui atributos, apenas métodos.

Métodos Abstratos

São métodos que não vão ser desenvolvidos na interface, apenas indicar que vai existir.

<<Interface>>

**Controlador**

+ Ligar ();

+ Desligar ();

+ abrirMenu ();

+ fecharMenu ();

+ maisVolume ();

+ menosVolume ();

+ ligarMudo ();

+ desligarMudo ();

+ play ();

+ pause ();

­

- Volume

- Ligado

- Tocando

**Controle Remoto**

Os gets e sets foram definidos como privados apenas para um exemplo.

O que acontece quando se encapsula

Todos os atributos podem ser privados ou protegidos, mas nunca públicos. Assim protegendo a parte interna do objeto.

Utilização

A classe Controle Remoto vai implementar o Controlador, ou seja, a classe Controle Remoto vai ter todos os métodos abstrados.

Representando por uma seta pontilhada. Porém nós iremos representar uma seta preenchida.

+ Ligar ();

+ Desligar ();

+ abrirMenu ();

+ fecharMenu ();

+ maisVolume ();

+ menosVolume ();

+ ligarMudo ();

+ desligarMudo ();

+ play ();

+ pause ();

- setVolume ();

- getVolume ();

- setLigado();

- getLigado()

- setTocando();

getTocando();

Interface Controlador

//Metodos 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 deligarMudo();

publico abstrato Metodo play();

publico abstrato Metodo pause();

FimInterface;

Classe ControleRemoto Implementa Controlador

//Obrigado sobrescrever os métodos da interface controlador

//Atributos

privado volume: inteiro;

privado ligado: logico;

privado tocando: logico;

//Metodos

publico Metodo Construtor ():

volume = 50;

ligado = falso;

tocando = falso;

FimMetodo;

privado Metodo getVolume ():

retorne this.volume;

FimMetodo;

privado Metodo getLigado ():

retorne this.ligado;

FimMetodo;

privado Metodo getTocando ():

retorne this.tocando;

FimMetodo;

privado Metodo setVolume (v : Inteiro):

this.volume = v;

FimMetodo;

privado Metodo setLigado (l : logico)

this.ligado = l;

FimMetodo;

privado Metodo setTocando (t : logico)

this.tocando = t;

FimMetodo;

//Sobrescrevendo métodos

publico Metodo ligar()

this.setLigado(Verdadeiro) ;

FimMetodo;

publico Metodo desligar()

this.setLigado(Falso) ;

FimMetodo;

publico Metodo abrirMenu()

escreva(getLigado());

escreva(getVolume());

para i = 0 ate getVolume() passo 10 faca

escreva (“|“);

FimPara

escreva(getTocando());

FimMetodo;

publico Metodo fecharMenu()

escreva (“Fechando menu”);

FimMetodo;

publico Metodo maisVolume()

se (getLigada()) entao

setVolume(getVolume() + 1) ;

FimMetodo;

publico Metodo menosVolume()

se (getLigada()) entao

setVolume(getVolume() - 1) ;

FimMetodo;

publico Metodo ligarMudo();

se ((getLigada()) e (getVolume > 0)) entao

setVolume(0) ;

FimMetodo;

publico Metodo deligarMudo();

se ((getLigada()) e (getVolume = 0)) entao

setVolume(50) ;

FimMetodo;

publico Metodo play();

se ((getLigada()) e (nao(getTocando()))) entao

setTocando(verdadeiro);

FimMetodo;

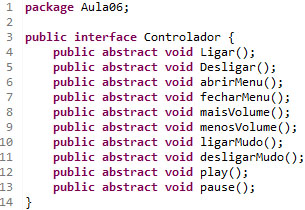
publico Metodo pause();

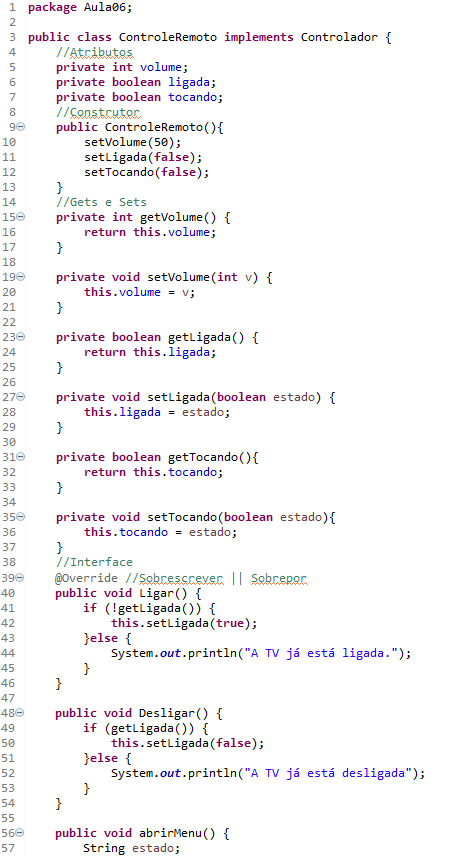
se ((getLigada()) e (getTocando())) entao

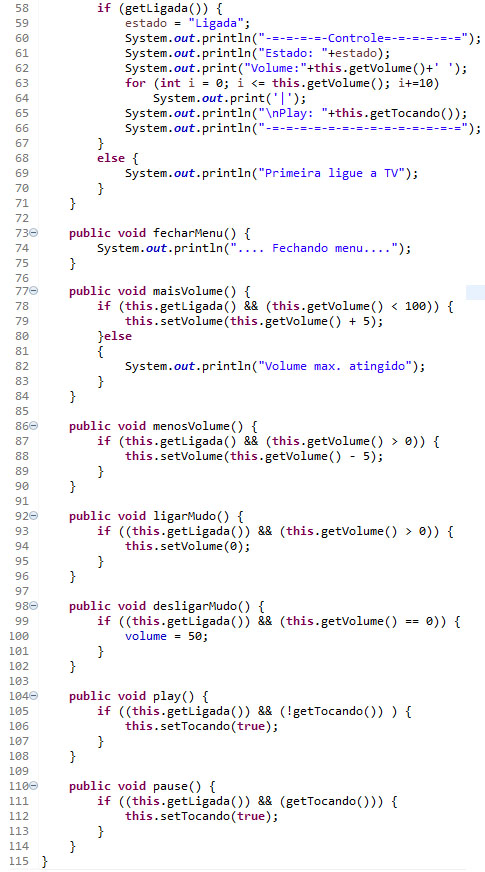
setTocando(falso);

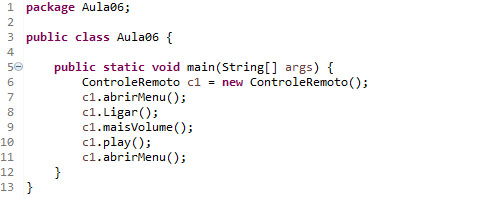
FimMetodo;

FimClasse;





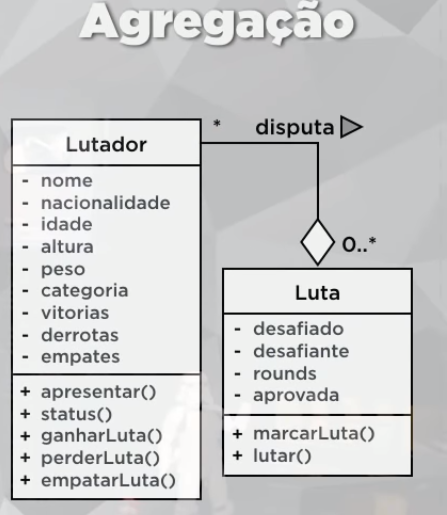




Relacionamento entre classes

Agregação

Conhecida como tem um. “A classe B tem um atributo da classe A”.



Luta agrega lutador (losango na ponta). (desafiado, desafiante).

Papel

Disputa. Todo relacionamento tem um papel.

Sentido

Seta que define um sentido.

Lutado disputa luta.

Multiplicidade

Cada lutador por participar de 0..\* lutas.

Cada luta existe com \* lutadores.

“Luta tem um lutador”. (desafiado, desafiante).

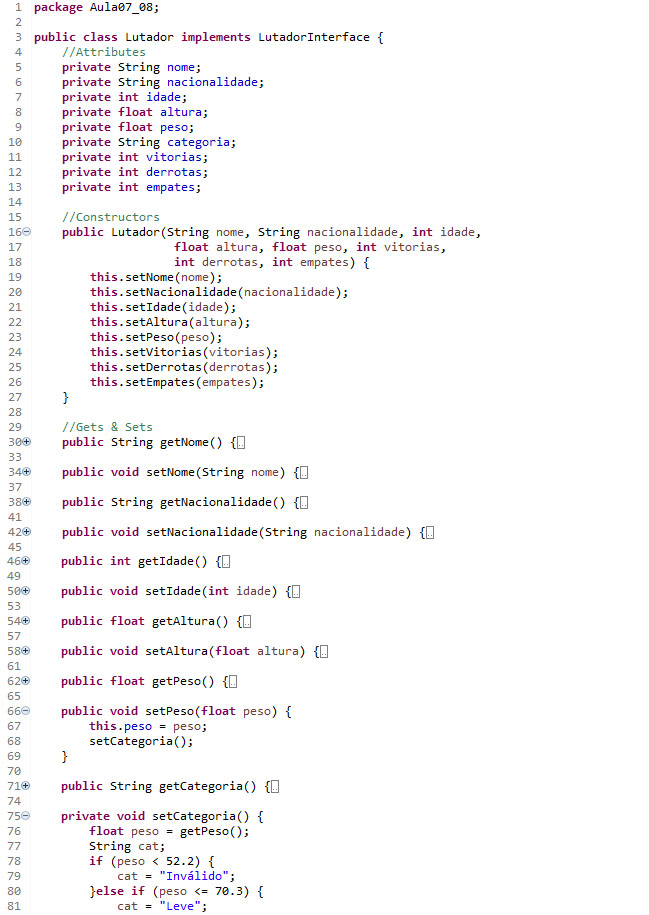
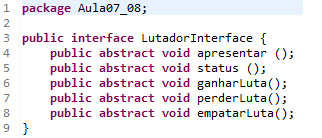
Obs: “É uma evolução do modelo de entidade relacionamento de banco de dados.”

Diagramas de relacionamentos só contém dados.

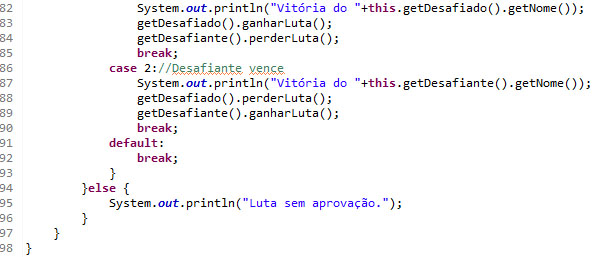
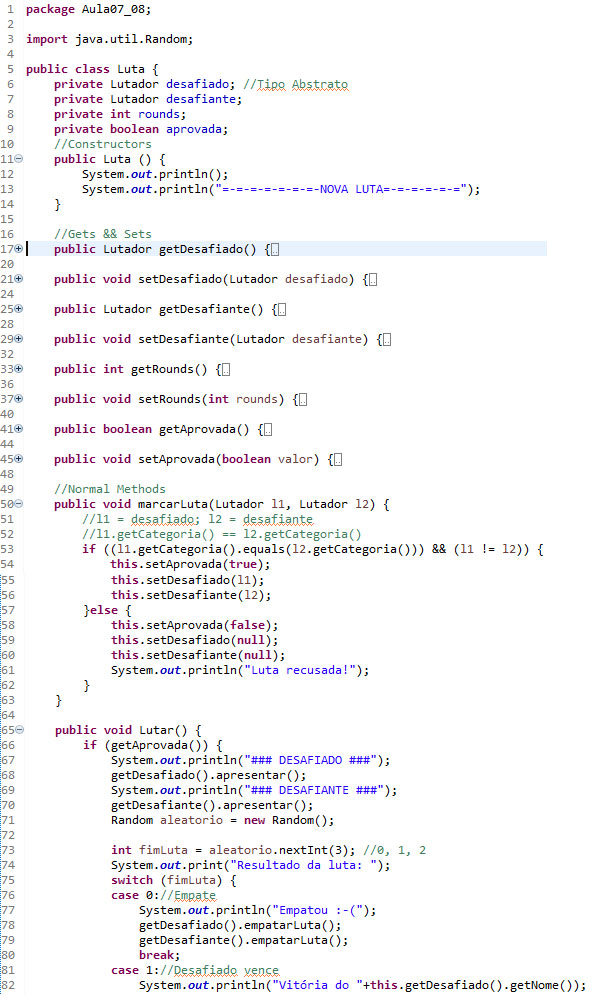
Diagramas de classes tem dados e funções.

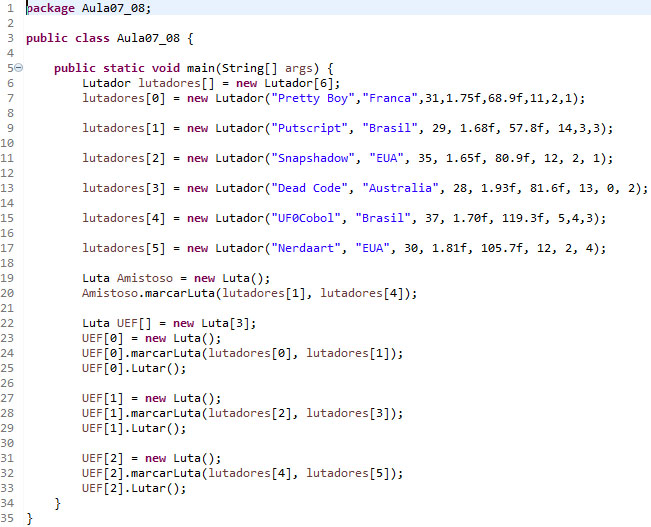
InterfaceLutador ► Lutador ► InterfaceLuta• ► Luta ► Main.

• : Não foi criado o código por questões de espaço, mas o ideal é que seja criado.







**

Exercícios

Teórico

1. (CAP-PD 2006) Uma casa está para uma planta arquitetônica assim como um objeto está para:
   1. Um método.
   2. Uma propriedade.
   3. Uma classe.
   4. Um atributo.
2. (EAGS 2015) A programação orientada a objetos se preocupa em produzir software que tenha as seguintes características:
   1. Natural, confiável, reutilizável, manutenível, extensível e oportuno.
   2. Natural, mensurável, manutenível, instantâneo, acessível e confiável.
   3. Confiável, reutilizável, extensível, mensurável, acessível e fácil de usar.
   4. Extensível, natural, mensurável, reutilizável, instantâneo e simples
3. (EAGS 2009) Assinale a alternativa incorreta com relação ao conceito de classe e objeto em Programação Orientada a Objetos.
   1. Uma classe é instância de um objeto.
   2. Um objeto é uma construção de software que encapsula estado e comportamento.
   3. Uma classe define os atributos e comportamentos compartilhados por um tipo de objeto.
   4. Em uma linguagem POO pura, tudo é um objeto, desde os tipos mais básicos.
4. (EAGS 2012) Correlacione a coluna da esquerda com a coluna da direita e assinale a alternativa que contenha a sequencia correta

1. Atributo
2. Classe
3. Comportamento
4. Domínio
5. Objeto

( ) Ação executada por um objeto.

( ) Define os atributos e comportamentos comuns.

( ) Características de uma classe que é visível.

( ) Construção de software que encapsula estado e comportamento.

( ) Estado onde o problema reside

1. 3, 2, 4, 5, 1.
2. 3, 2, 1, 5, 4.
3. 2, 3, 1, 5, 4.
4. 2, 4, 3, 1, 5.
5. (EAGS 2009) Assinale a alternativa incorreta sobre a definição de termos usados em POO.
   1. Uma variável interna é um valor mantido dentro do objeto.
   2. Atributos são as características de uma classe visíveis externamente.
   3. Comportamentos são as características de uma classe invisível externamente.
   4. O estado de um objeto é o significado combinando das variáveis internas do objeto.
6. (EAGS 2013) Associe as colunas.
7. Construtor
8. Acessor
9. Mutante - Modificadores
10. Tipos

( ) Métodos que dão acesso aos dados internos.

( ) Define as diferentes espécies da valores que podem ser usados

( ) Método que permitem que se altere o estado do objeto

( ) Método para inicializar objetos durante a instanciação.

1. 1, 2, 3, 4.
2. 2, 4, 1, 3.
3. 2, 4, 3, 1.
4. 4, 2, 3, 1.
5. (EAGS 2014) Quais são os três pilares da POO?
6. Herança, polimorfismo e instância.
7. Herança, polimorfismo e metodologia.
8. Encapsulamento, herança e polimorfismo.
9. Encapsulamento, polimorfismo e metodologia.
10. (EAGS 2013) Correlacione a coluna superior com a inferior.
    1. Classe
    2. Interface
    3. Construtor
    4. Objeto

( ) Define o que uma entidade pode fazer com o objeto.

( ) Instanciação de uma classe.

( ) Define os atributos e comportamentos.

( ) Utilizada para inicializar objetos.

1. 1, 2, 4, 3.
2. 1, 4, 2, 3.
3. 2, 3, 1, 4.
4. 2, 4, 1, 3.
5. (EAGS 2010) A maioria das linguagens orientadas a objetos suporta quais níveis de acesso?
   1. Secreto, público e primário.
   2. Público, protegido e privado.
   3. Público, protegido e primário.
   4. Público, privado e secundário.
6. (EAGS 2011) No contexto nível de acesso, selecione a alternativa que contém o nível que garantes apenas acesso para aquele objeto e suas subclasses.
   1. Público.
   2. Protegido.
   3. Privado.
   4. Oculto.

Gabarito:

1. C.
2. A. – COMERN.
3. A.
4. B.
5. C.
6. C.
7. C.
8. D.
9. B.
10. B.

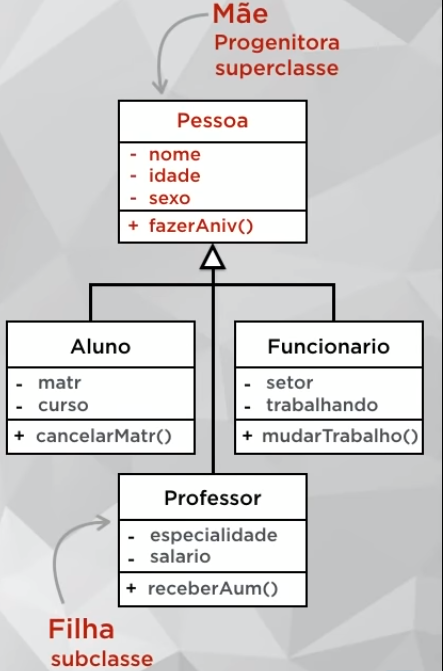
Pratica:



Herança

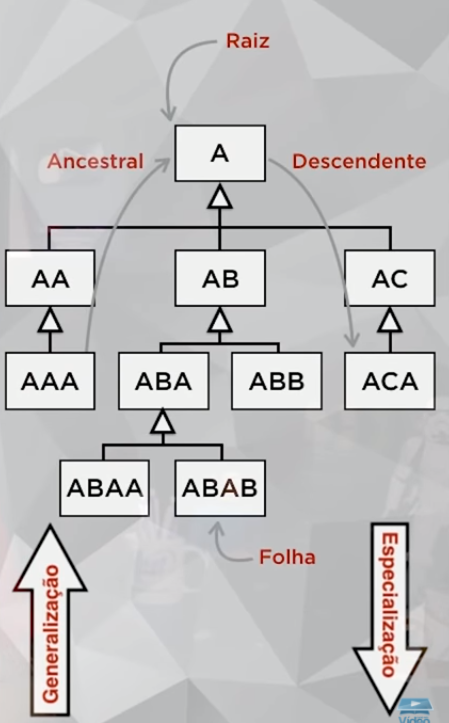
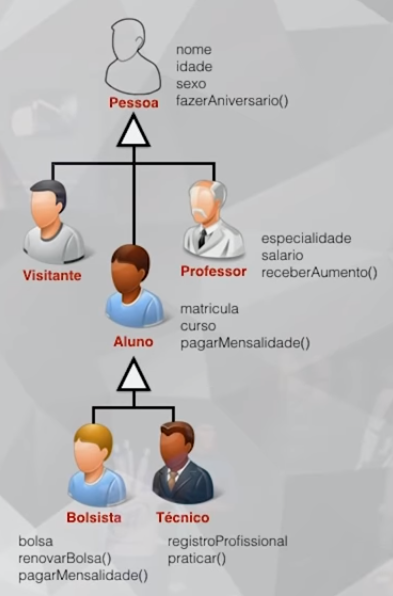
Permite basear uma nova classe na definição de uma outra classe previamente existente.

A herança será aplicada tanto para características, quanto para os comportamentos (atributos e métodos).



Aluno, funcionário e professor herdam de pessoas.

Ou seja, além dos métodos próprios, terão os métodos da classe Pessoa.

 Pessoa é do tipo abstrato.

Tipos de heranças:

Herança de Implementação

Herança pobre, é a mais simples que se tem, pois a mesma não implementa nada novo em relação a superclasse.

Herança para diferença

Implementa novos métodos ou sobrepõem algum existente, em relação a superclasse.

Tipo Abstrato e Final

Classe Abstrato

Não pode ser instanciada. Só pode servir como progenitora.

Método Abstrato

Declarado, mas não implementado na progenitora (Obs: só pode ser utilizado em uma interface ou classe abstrata).

Classe final

Não pode ser herdada por outra classe. Obrigatoriamente é uma folha (“Classe estéril”).

Método Final

Não pode ser sobescrito pelas subclasses. Obrigatoriamente herdado.

Conceito Polimorfismo

“Poli = muitas / Morfo = formas.

Muitas formas de se fazer uma coisa.”

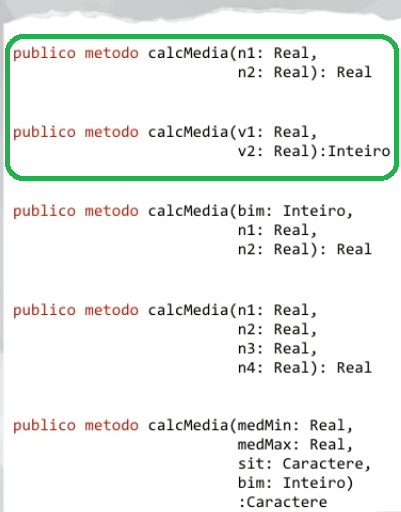
Permite que um mesmo nome represente vários comportamentos diferentes.

Assinatura

É a quantidade e tipos de parâmetros.

Todo método tem uma, e você vai ter que identificar essas assinaturas, e descobrir se são iguais, diferentes.

Os métodos contornados possuem a mesma assinatura.



Tipos Polimorfismo

Várias bibliografias citam vários tipos de polimorfismo, porém só será ensinado os dois principais nesse curso.

Sobreposição - Override

Acontece quando substituímos um método de superclasses na sua subclasse, usando a mesma assinatura.

Só acontece uma vez em cada classe, e tem que estar em classes diferentes, com a mesma assinatura.

Sobrecarga - Overload

Consiste basicamente em criar variações de um mesmo método, porém com assinaturas, e escopo diferentes.

Só acontece na mesma classe, e tem assinaturas diferentes.