**Apple Academy**

**Guilherme Dalosto**

**to do -** fazer questões de concurso sobre Orientação a Objeto e Java.

treinar no URI | Java e C.

Fazer todos exercícios do Jair e do Michel

**RESUMAO** <https://www.devmedia.com.br/como-criar-minha-primeira-classe-em-java/38940>

**SUMÁRIO**

https://www.devmedia.com.br/guia/linguagem-java/38169

unsigned => números positivos ou zero.

int - 4 bytes

ponteiro altera valores apontados para ele sem necessitar referencia-los de novo.

Another difference between Polymorphism and Abstraction is that Abstraction is implemented using [abstract class and interface in Java](http://java67.blogspot.sg/2012/09/what-is-difference-between-interface-abstract-class-java.html) while Polymorphism is supported by overloading and overriding in Java.

Read more: <http://www.java67.com/2015/05/difference-between-abstraction-and.html#ixzz5RNpDAHgx>

A [**associação**](https://en.wikipedia.org/wiki/Association_(object-oriented_programming)) entre dois objetos ocorre quando eles são completamente independentes entre si mas eventualmente está relacionados. Ela pode ser considerada uma relação de muitos para muitos. Não há propriedade nem dependência entre eles. A relação é eventual.

A [**agregação**](https://en.wikipedia.org/wiki/Object_composition#Aggregation) não deixa de ser uma associação mas existe uma exclusividade e determinados objetos só podem se relacionar a um objeto específico. É uma relação de um para muitos. Um objeto é proprietário de outros mas não há dependência, então ambos podem existir mesmo que a relação não se estabeleça. Na verdade há controvérsias sobre a exata definição e o que é mais importante, a relação de um para muitos ou a propriedade.

Um exemplo é a relação entre os professores e os departamentos. Departamentos podem ter vários professores. E o professor só pode estar vinculado a um departamento. Mas eles são independentes. Um professor pode existir sem vínculo com um departamento e não depende de professores para existir.

A [**composição**](https://en.wikipedia.org/wiki/Object_composition) é uma agregação que possui dependência entre os objetos, ou seja, se o objeto principal for destruído, os objetos que o compõem não podem existir mais. Há a chamada relação de morte.

**método init** serve para inicializar qualquer processo, é um método que faz a execução de qualquer processo dado. Fazer inicialização de um programa Applet

[**Singleton**](https://www.oodesign.com/singleton-pattern.html) - **Ensure that only one instance of a class is created** and **Provide a global access point to the object.**

**Conceitos de Orientação a Objeto**

Objeto

Mensagem

Classe

Abstração

Encapsulamento

Herança

Polimorfismo

**Vantagens de Orientação a Objeto**

o aumento de produtividade

o reuso de código

a redução das linhas de código programadas

a separação de responsabilidades

a componentização

a maior flexibilidade do sistema

a escalabilidade

a facilidade na manutenção, dentre outras vantagens.

**Orientação a Objeto** envolve conceitos de abstração, encapsulamento, generalização e especialização.

**tipos de variáveis**

**Variáveis Parâmetros:** são variáveis que as funções necessitam para que possam executar alguma determinada ação.

**Variáveis Globais:** são variáveis criadas logo após a criação da classe e essa variáveis podem ser utilizadas em qualquer função dessa classe, caso essa variável seja public poderá ser utilizada em outras classes também.

**Variáveis Locais:** são variáveis criadas dentro de uma função e esta somente será acessível dentro da função, sendo assim após o final da função essa variável deixará de existir.

**Variáveis Objetos:** como já estudado, são variáveis criadas a partir de classes, e essas variáveis passam a ter acesso a todos os métodos da classes a partir qual foi criada, essa variáveis poderão assumir a forma de: Variáveis Objeto Parâmetros, Variáveis Objeto Global, Variáveis Objeto Local.

**Apontador This**

O apontador “this”, é utilizado para diferenciar uma variável global de uma variável local basicamente, pois durante a codificação poderemos utilizar variáveis com a mesma nomenclatura.

**Classes** - Conceito estático

Atributos são características da classe

**Atributos**

Atributos de instância são criados para cada objeto, e não são compartilhados entre eles.

Atributos de classe são criados de uma vez só, para a classe e é compartilhado para todas as instâncias da mesma ( podem ser acessados diretamente de outras classes ).

**Atributo estático**

É único e compartilhado entre todas as instâncias da classe.

Métodos são ações da classe

**Objetos** - Conceito dinâmico

Objeto é o que é criado a partir da classe.

São instâncias das classes

Uma classe pode ser usada para instanciar vários objetos.

Objetos que são instâncias da mesma classe, possuem os mesmos métodos e atributos, e são

independentes entre si.

É impossível ter um objeto sem uma classe que o defina.

**Mensagens**

Executa o método

Composta por três partes **->** Objeto destino que receberá a mensagem, nome do método a ser invocado pela mensagem e parâmetros necessários.

Há três maneiras de chamar um método:

Na mesma classe

Referência a um objeto

Métodos estáticos

**Construtor**

Método que inicializa o objeto, executado na instanciação como ‘new’

Deve ter o mesmo número da classe

Caso não inicializado, em java, são implementados padronizadamente com tipos números : 0, Valores booleanos : false e referências : null.

**Conceitos iniciais**

**Objeto**

Identidade, atributo, estado e comportamento.

Conceitos reais -> carro, animal, barco e etc.

Conceitos abstratos -> conta poupança, funcionário, pessoa física e etc.

Instâncias são as cópias do objeto (carro2, carro3).

**Abstração**

Representam os diversos pontos de vista sobre um determinado objeto.

Uma classe abstrata não pode ser instanciada

Define a interface com o mundo externo mas deixa o comportamento para ser resolvido adiante.

Determinando apenas os aspectos relevantes para sua chamada.

Abstração é o ato de resumir características ou ações similares partilhadas entre dois ou mais objetos

*Como um gato pode ser visto pelo dono como mascote e pelo veterinário por um ser dotado de sistema esquelético, respiratório e etc.*

**Classificação**

Agrupar os objetos que possuam os mesmos atributos.

*Como veículos podem ser agrupados por cor e número de rodas.*

**PILAR - ENCAPSULAMENTO**

**Encapsulamento**

Restringe a visibilidade da informação para obter maior clareza, ajudando na manutenção e na reutilização dos códigos.

Cada módulo deve obter apenas o que lhe diz respeito.

Deixar visível apenas o necessário.

Qualquer classe encapsulada deverá proteger seus atributos não permitindo que elementos externos a ela alterem seus valores, necessário então, deixar esses atributos privados.

Se necessário a alteração de um valor, usa-se GETTER ( pegar o valor) e SETTER ( alterar o valor )

As funcionalidades da classe ficam separadas de sua implementação, existe um “contrato”, de cada classe com o mundo externo, chamado de interface

**Acoplamento**

Refere-se ao nível que uma classe conhece e usa os membros de outra classe, para reduzir danos em suas alterações.

**Coesão**

Refere-se a responsabilidade de uma classe, com atributos e métodos para fazerem apenas sua funcionalidade.

***Coesão está, na verdade, ligado ao princípio da responsabilidade única, diz que uma classe deve ter apenas uma única responsabilidade e realizá-la de maneira satisfatória, ou seja, uma classe não deve assumir responsabilidades que não são suas . Já o acoplamento significa o quanto uma classe depende da outra para funcionar. E quanto maior for esta dependência entre ambas, dizemos que estas classes elas estão fortemente acopladas.***

**Um bom software deve ter alta Coesão e baixo Acoplamento.**

*Como uma calculadora e um carro, podem ser olhados e utilizá-los sem abri-los para ver seu funcionamento.*

PILAR - HERANÇA

**Herança**

Usa-se o termo extends.

Relacionado ao conceito, **é um(a).**

**Generalização**

Quando partimos de uma classe e chegamos a sua classe mãe.

**Especialização**

Quando partimos de uma classe mãe e chegamos as classes filhas.

Gato é uma especialização de Animal e Animal é uma generalização de Gato. Gato estende animal.

Segue o princípio da visibilidade, o que é privado em uma classe não pode ser observado em suas subclasses.

Os objetos podem ser subdivididos e as subclasses “herdam” os mesmos atributos da superclasse, auxiliando na reutilização do código.

Possui atributos e métodos gerais.

**Superclasse ->** Classe principal que fornece os atributos e métodos em comum.

**Subclasse ->** Classe que herda os atributos e métodos.

Em Java só é permitido heranças simples ( uma subclasse só pode ter uma superclasse).

*Como os meios de transporte, possuem características em comum e podem ser reutilizados a partir da herança.*

***Composição -> Conceito, tem um (a).***

**PILAR - POLIMORFISMO**

**Polimorfismo**

Entendido como um código que pode possuir diferentes comportamentos, utilizando o mesmo código por mais de uma classe.

Permite programar de forma geral em vez de específica.

Dependente da Herança.  
Pode ocorrer a redefinição de métodos e a sobrecarga deles, usando override e overloading.

**Override** (run-time polymorphism) -> Dois métodos têm o mesmo nome e os mesmos parâmetros, um usa na superclasse e outro nas subclasses.

**Overload** (compile-time polymorphism) -> Dois métodos têm o mesmo nome mas parâmetros diferentes na mesma classe.

**Existem 3 tipos de Polimorfismo que a linguagem pode ter, classificados em “Universal e Ad-Hoc”**

Universal:

o Inclusão: um ponteiro para classe mãe pode apontar para uma instância de

uma classe filha (tipo de polimorfismo mais básico que existe).

o Paramétrico: se restringe ao uso de templates (C++, por exemplo) e

generics (Java).

Ad-Hoc:

o Sobrecarga: duas funções/métodos com o mesmo nome, mas assinaturas

diferentes.

A **sobrescrita (ou *override*)** está diretamente relacionada à orientação a objetos, mais especificamente com a herança. Com a sobrescrita, conseguimos especializar os métodos herdados das superclasses, alterando o seu comportamento nas subclasses por um mais específico.

A **sobrecarga de métodos (overload)** é um conceito do polimorfismo que consiste basicamente em criar variações de um mesmo método, ou seja, a criação de dois ou mais métodos com nomes totalmente iguais em uma classe. A Sobrecarga permite que utilizemos o mesmo nome em mais de um método contanto que suas listas de argumentos sejam diferentes para que seja feita a separação dos mesmos.

*Como análise a classe Polígonos, que representa as formas geométricas, o método para calcular a área deve ter diferentes comportamentos dependendo do polígono em questão.*

**Persistência**

O estado e o comportamento de um objeto variam com o tempo.

*Como o preço de um produto.*

***Modificadores de acesso***

***Público ->*** Atributos e métodos são acessíveis em todos os métodos de todas as classes. ( Nível menos rígido de encapsulamento )

***Privado ->*** Atributos e métodos são acessíveis somente nos métodos da própria classe. ( Nível mais rígido de encapsulamento )

***Protegido ->*** Atributos e métodos são acessíveis nos métodos da própria classe e suas subclasses.

***Nada especificado, equivale a “pacote” ou “default” ou “friendly” ->*** Atributos e métodos são acessíveis somente nos métodos das classes que pertencem ao “pacote” em que foram criados.

**Ordem de restrição**

Privado - Default - Protegido - Público



**Classes abstratas**

Classes abstratas são classes que, basicamente, definem um tipo, digo, elas são apenas um rascunho de como as classes que herdarem dela devem se comportar.

Neste caso, qualquer classe que herdar de Animal deverá implementar o método getHabitat(), caso contrário ocorrerá um erro de compilação. Quanto ao método getRaca(), as classes filhas podem simplesmente usar o código que já existe na classe abstrata, sem se preocupar em implementá-lo.

**Não podem ser instanciadas**

Sempre serão estendidas, quase sempre usará heranças.

Representam conceitos e são utilizadas para criar uma hierarquia de classes.

Devem ser sobrescritos nas subclasses. Se a subclasse não sobrescrever todos os métodos, ela deve ser declarada como abstract.

Uma classe que contenha algum método abstrato deve ser declarada como abstrata.

Não necessariamente contém métodos abstratos, podem apenas

representar conceitos e serem a base para uma hierarquia de classes.

**Interfaces**

**A Interface é muito utilizada em grandes projetos para obrigar o programador a seguir o padrão do projeto, por esta tratar-se de um contrato onde o mesmo é obrigado a implementar seus métodos, ele deverá sempre seguir o padrão de implementação da Interface.**

**Ela só expõe o que o objeto deve fazer, e não como ele faz, nem o que ele tem. Como ele faz vai ser definido**

**Pode-se ter multiplas interfaces - IMPLEMENTS**

Permite definir comportamentos comuns à várias classes que não tenham relações hierárquicas entre si.

**IMPORTANTE**



**Métodos get e set**

Concentra toda a validação e exposição de um atributo em um único lugar.

**QUESTÕES**

Suponha a existência das classes A, B e C, sendo que A estende B, a qual estende C. Sendo assim, A em relação à C é um exemplo de \_\_especializacao\_\_\_\_\_\_\_, enquanto o inverso é um exemplo de \_\_\_\_generalizacao\_\_\_\_\_. Se A possui 3 versões de um mesmo método (mesmo nome), porém cada um com uma assinatura própria, temos um exemplo de \_\_\_\_polimorfismo\_\_\_\_\_. Se um determinado atributo de B é \_protegido\_\_\_\_\_\_\_\_ ele será visível tanto para B quanto para A.”

Possibilidade de ter mais de um método com o mesmo nome, na

mesma classe, mas com parâmetros distintos.

O conjunto: nome do método + parâmetros , é chamado de

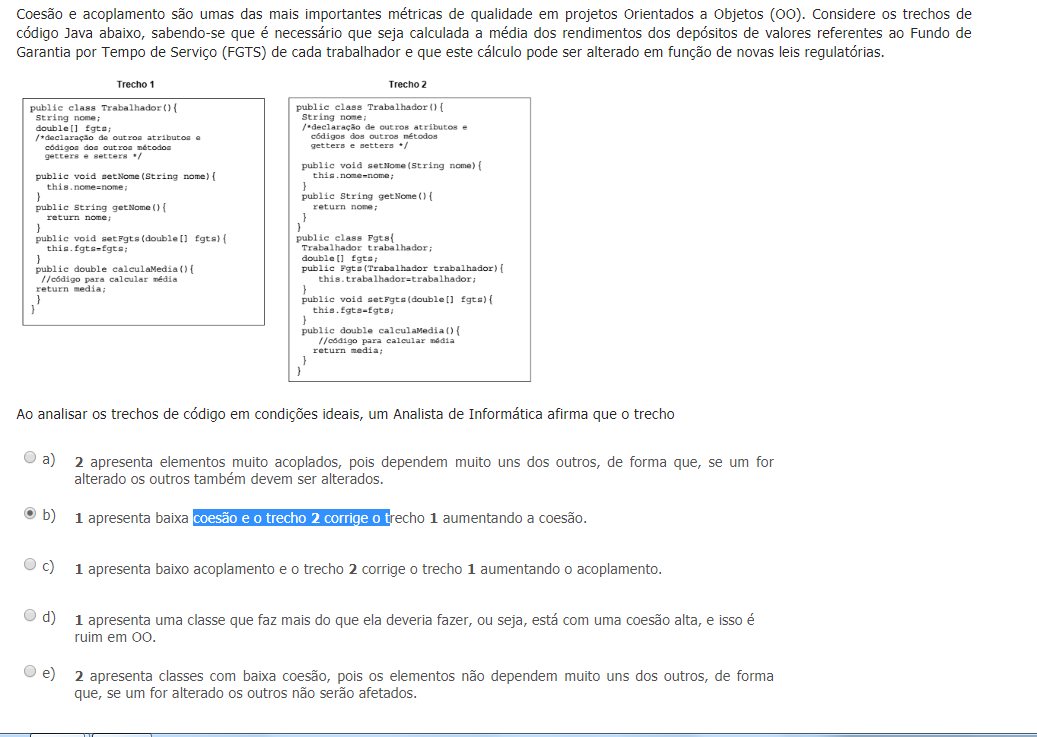
assinatura do método.

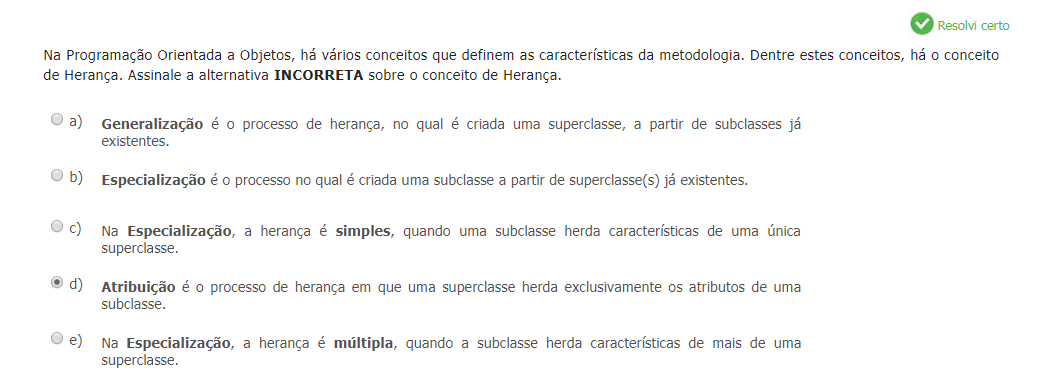
Em um bom projeto OO, as classes devem ter alta coesão e baixo acoplamento.

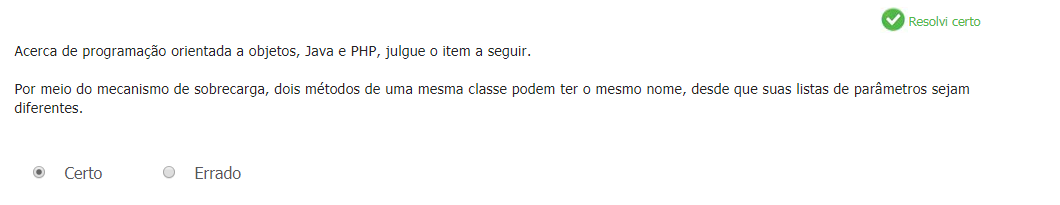
Não é permitido ter dois métodos na mesma classe declarados com a

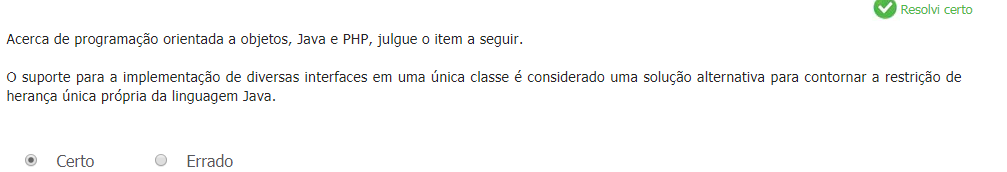
mesma assinatura.

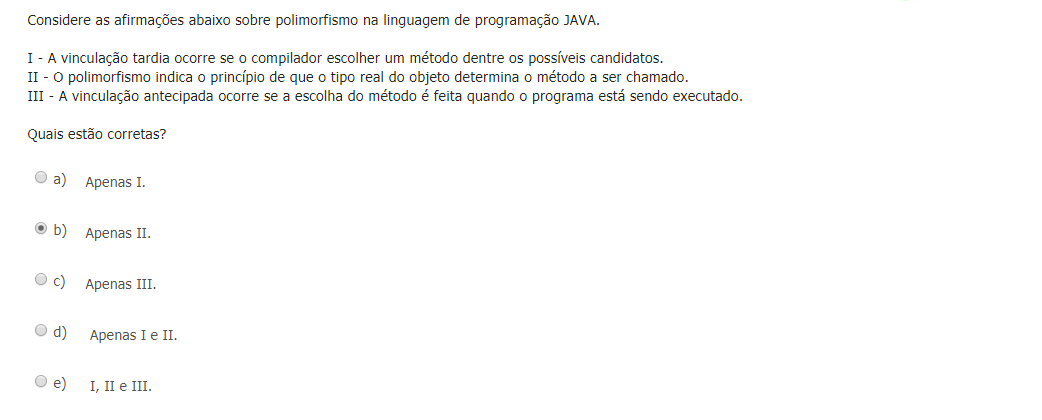
**Polimorfismo** é uma característica que reduz bastante o esforço necessário para ampliar o projeto de um sistema orientado a objetos, permitindo que várias operações diferentes tenham o mesmo nome.









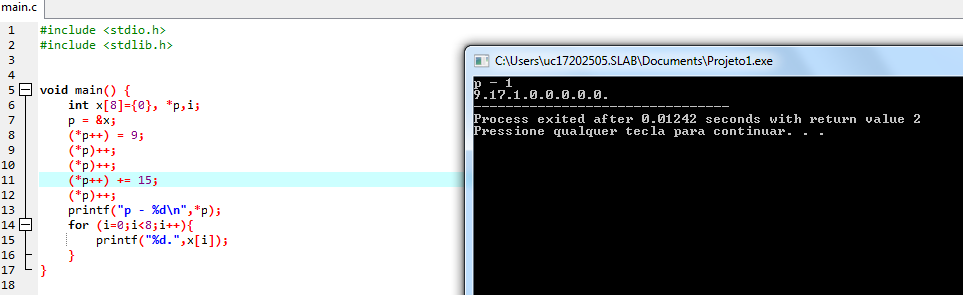


<https://github.com/ythecombinator/bepid-studies/blob/master/material/linguagem-C/exercicios/ifce.md>

https://github.com/ythecombinator/bepid-studies

**PONTEIRO**

Mexer no ponteiro também mexe na variável em que ela está apontada.

Igualar uma variável a um ponteiro e alterá-la, não fará qualquer efeito nela.

Existem ponteiros de ponteiros em C

