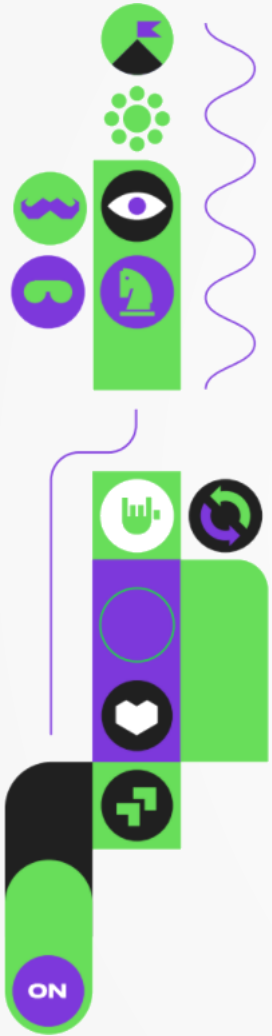


POO - Programação orientada a objetos

GAMA ACADEMY

Programação orientada a objetos

A programação orientada a **objetos surgiu como uma alternativa a características da programação estruturada.** O intuito da sua criação também foi o de **aproximar o manuseio das estruturas de um programa ao manuseio das coisas do mundo real**, daí o nome "objeto" como uma algo genérico, que pode representar qualquer coisa tangível.

Esse paradigma se baseia principalmente em dois conceitos chave: *classes* e *objetos*. Todos os outros conceitos, igualmente importantes, são construídos em cima desses dois.

Alan Curtis Kay - Conhecido por ser um dos criadores da POO e da linguagem de programação Smalltalk.

Graduado em matemática e biologia molecular pela Universidade do Colorado. Com seus conhecimentos em Biologia e Matemática, formulou sua "analogia algébrico-biológica" e lançou o postulado de que o computador ideal deveria: funcionar como um organismo vivo, isto é, cada "célula" comportar-se-ia relacionando-se com outras a fim de alcançar um objetivo, contudo, funcionando de forma autônoma. As células poderiam também reagrupar-se para resolver um outro problema ou desempenhar outras funções.

Alan Kay pensou em como construir um sistema de software a partir de agentes autônomos que interagem entre si, estabelecendo os seguintes princípios:

+ Qualquer coisa, que pode ser percebida pelos sentidos e descrita por suas características é um objeto.

+ Objetos realizam tarefas através da requisição de serviços.

+ Cada objeto pertence a uma determinada classe.

+ Uma classe agrupa objetos similares.

+ Uma classe possui comportamentos associados a o objeto.

+ Classes são organizadas em hierarquias.

Confiável	O isolamento entre as parte gera software seguro. Ao tentar alterar uma parte, nenhuma outra é afetada.
Oportuno	Ao dividir em parte, várias delas podem ser desenvolvida em paralelo.
Manutenível	Atualizar um software é mais fácil. Uma pequena modificação vai beneficiar todas as partes que usarem o objeto.
Extensível	O software não é estático. Ele deve crescer para permanecer útil.
Reutilizável	Um objeto deve ser estruturado a ponto de ser possível a reutilização em outros sistemas futuros.
Natural	Mais fácil de entender. Você se preocupa mais na funcionalidade do que nos detalhes de implementação.

O que é OBJETO?

- Coisas materiais ou abstratas que podem ser percebidas pelos sentidos e descritas por meio das suas características, comportamentos e estado atual.

- É a instância de uma classe.

POO - Objeto

Coisas materiais ou abstratas que podem ser percebidas pelos sentidos e descritas por meio das suas características, comportamentos e estado atual. É a instância de uma classe.

POO - Objeto

Coisas materiais ou abstratas que podem ser percebidas pelos sentidos e descritas por meio das suas características, comportamentos e estado atual. É a instância de uma classe.

Características

cor: cinza
categoria: sedan
fabricante: GM
modelo: prisma

Comportamento

ligar / desligar
Se mover

Estado

novo / conservado
parado
abastecido

Classe

Classe é uma descrição que abstrai um conjunto de objetos com características similares. Define atributos e métodos comuns que serão compartilhados por um objeto.

01. Coisas que eu tenho:

(atributos)
modelo
cor
categoria (sedan / conversível / SUV)
nivel combustivel

02. Coisas que eu faço:

(métodos)
Mover
Freiar
Abrir as portas

03. Como estou atualmente?

(estado)
prisma
prata
sedan
50% do tanque abastecido

Classe

Objeto

Orientação a objetos

Abstração

Trata-se da capacidade de filtrar apenas os dados relevantes e necessários para criação do nosso objeto. Trazer o cenário real de modo *simplificado* e *objetivo* para sua aplicação/objeto.

classe Cliente
nome
cor do cabelo
nome do pet
cpf
comida favorita
email

classe ClienteBanco
nome
cpf
email

classe ClientePetShop
nome
nome do pet
endereço

Pilares da Orientação a objetos

Encapsulamento

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

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

Objetos bem encapsulados produzem padrões e geram proteção aos seus produtos e aos seus usuários.

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.

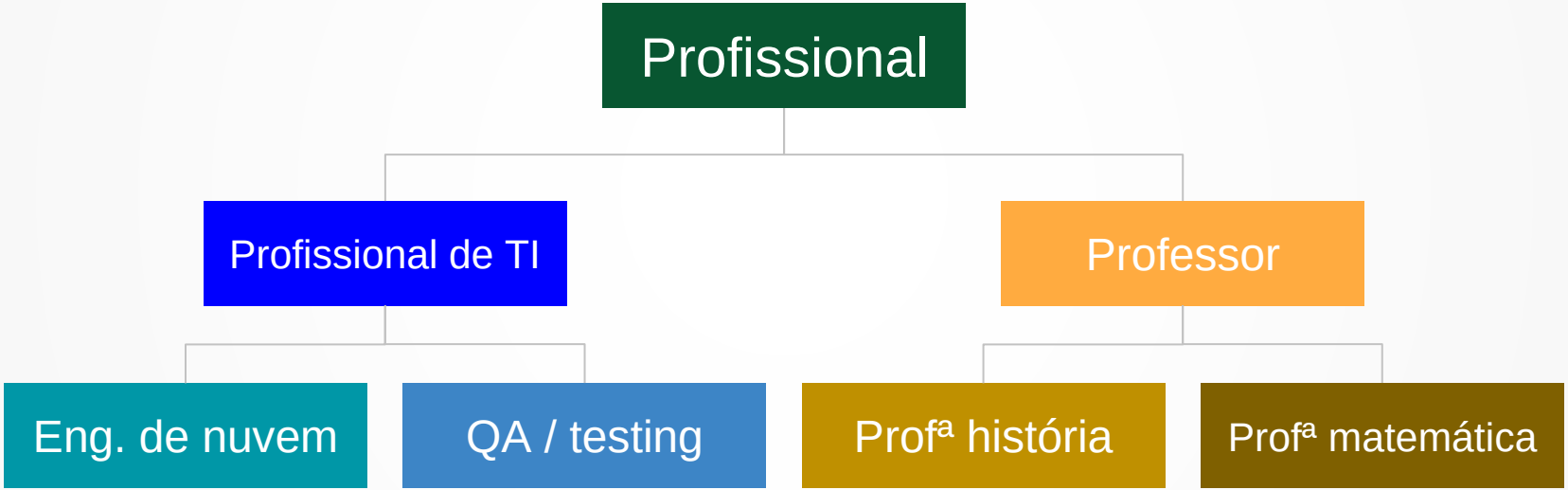
Benefícios do Encapsulamento

Modularidade	Objeto pode ser escrito e mantido independentemente de outros objetos, o que permite que cada objeto seja utilizado livremente no sistema.
Ocultação de informações	Mesmo com a possibilidade de comunicação via interface, o objeto pode manter informações privadas e métodos podem ser modificados em qualquer momento sem afetar os outros objetos que dependem dele.
Facilitar reutilização de códigos	Uma Classe bem encapsulada é possível a reutilização até em outros softwares

Pilares da Orientação a objetos

Herança

Herança é a capacidade de uma subclasse, de ter acesso às propriedades da superclasse, a ela relacionada. Ela permite basear uma nova classe na definição de uma outra classe previamente existente.


Profissional
Profissional de TI
Professor
Eng. de nuvem
QA / testing
Profª história
Profª matemática

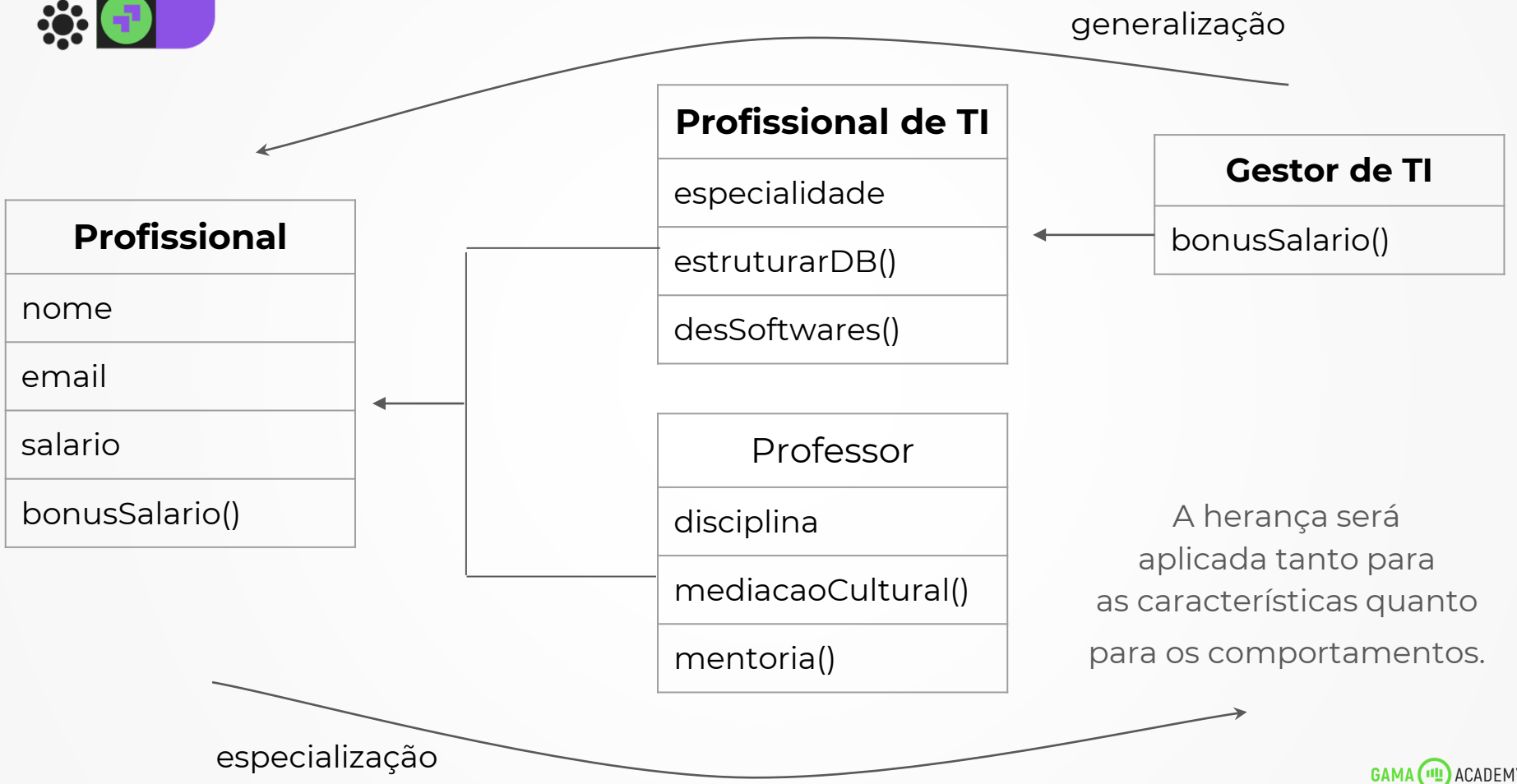
Tipos de herança:

Herança de implementação: Utiliza características e comportamentos da superclasse ou das classes ancestrais, porém não implemente nenhum novo método.

Herança para diferença: Além de utilizar características e comportamentos da superclasse, implementa novos atributos e/ou procedimentos.

Profissional de TI
nome
email
especialidade
salario
bonusSalario()
estruturarDB()
desSoftwares()

Professor
nome
email
disciplina
salario
bonusSalario()
mediacaoCultural()
mentoria()


generalização
Profissional
nome
email
salario
bonusSalario()
Profissional de TI
especialidade
estruturarDB()
desSoftwares()
Gestor de TI
bonusSalario()
Professor
disciplina
mediacaoCultural()
mentoria()
A herança será aplicada tanto para as características quanto para os comportamentos.
especialização

Extends

A palavra-chave *extends* em Java indica que a classe filha herda ou adquire as propriedades da classe pai. Esta palavra-chave basicamente estabelece uma relação de *herança* entre classes.

Se uma classe estende outra classe, dizemos que ela adquiriu todas as propriedades e comportamento da classe pai.

Usamos a palavra-chave *extends* em Java entre dois nomes de classe que queremos conectar no relacionamento de herança.

```
public class MinhaClasse extends OutraClasse {

}
```

Interfaces

Interface é um modelo, muito semelhante a classe Java, mas a única diferença é que possui métodos abstratos e constantes estáticas.

A interface deve ser "implementada" (como herdada) por outra classe com a palavra- *implements.*

O principal objetivo de uma interface é obter segurança, ocultando certos detalhes e exibindo apenas os detalhes importantes de um objeto

1. As interfaces não podem ser usadas para criar objetos
2. Métodos de interface não possuem corpo
3. Os métodos de interface são por padrão abstracte public
4. Uma interface não pode conter um construtor (já que não pode ser usada para criar objetos)

Implements

A palavra-chave implements é útil quando queremos usar uma interface na classe. Resumindo, a palavra-chave implements é útil para implementar as interfaces nas classes.

Sabemos que uma interface em Java é definida como um tipo especial de classe que contém apenas métodos abstratos dentro dela; os métodos que não possuem nenhuma implementação ou corpo dentro deles. Ele apenas fornece um plano, contrato ou um protótipo que as classes devem seguir.

A palavra-chave implements é usada por uma classe para que ela possa seguir ou aderir ao contrato fornecido pela interface.

```
public class MinhaClasse implements umaInterface {
}
```

A classe que implementa a interface deve fornecer a implementação concreta de todos os métodos declarados na interface, fornecendo o corpo do método aos métodos.

Pilares da Orientação a objetos

Polimorfismo

POLI = muitas
MORFO = formas = Muitas formas de se fazer alguma coisa

Permite que classes pertencentes a uma mesma linha de herança possuam comportamentos diferentes para o mesmo método.

Exemplo: em um jogo de xadrez temos várias peças, cada peça se movimenta, porém cada tipo de peça se movimenta de uma maneira diferente

(o peão vai pra frente, o cavalo anda em L, o bispo em diagonal e assim por diante)

• Desta maneira, todos (peão, cavalo e bispo) são peças, porém cada um se movimenta de uma maneira diferente!

O polimorfismo em Java se manifesta apenas na chamada de métodos

• Então ao passar uma mensagem para um objeto peça, dizendo para ele se mover, o Java identifica qual o tipo de peça é e fará o movimento de acordo com o tipo

```
public abstract class Peca {
        public abstract void
mover();
}
```

```
public class  Peao extends Peca {
        @Override
        public  void mover() {
                System.out.println("Andar
para frente");
        }
}
```

```
public class  Peao extends Peca {
        @Override
        public  void mover() {
                System.out.println("Andar para frente");
        }
}
```

```
public abstract interface Peca {
        public abstract void mover();
}
```

```
public class  Cavalo extends Peca {
        @Override
        public  void mover() {
                System.out.println("Andar em "L"");
        }
}
```

```
public class Bispo extends Peca {
        @Override
        public  void mover() {
                System.out.println("Andar em diagonal");
        }
}
```

Exercício:

1. Crie uma estrutura de herança para demonstrar o polimorfismo utilizando classes de animais

1. Crie o método comunicar() e movimentar() na classe Animal

1. Nas classes descendentes de animais, sobrescreva os métodos citados de acordo com os tipos de animais

Obrigada

GAMA ACADEMY


GAMA
ACADEMY