
Orientação a Objetos
Encapsulamento
Herança
Composição
Polimorfismo
Princípio da abstração

Introdução

- A reutilização de código economiza tempo e reduz problemas depois que o sistema torna-se operacional.
- Como o objetivo principal da POO é a reutilização de código, novas técnicas surgem para garantir tal recurso.
- Herança e polimorfismos são consideradas as principais técnicas que garantem a reutilização de código.

Mas como isso de reutilização de código funciona ???

CTRL C + CTRL V ??

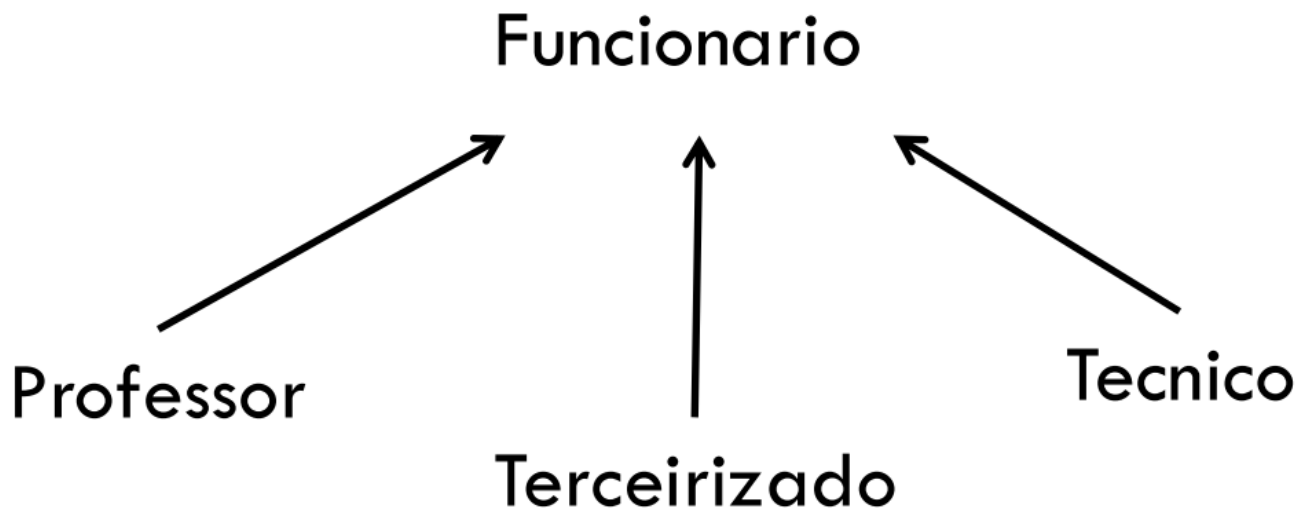
Herança

- Quando falamos a palavra herança, qual a primeira coisa que pensamos ??

Herança

- A herança é uma forma eficaz de reutilização de código.
- Criamos novas classes a partir de classes já existentes, absorvendo seus atributos e comportamentos.
- Com a utilização da herança criamos uma hierarquia de classes.
- Essas classes estão relacionadas, ou seja, possuem características semelhantes.
  - Ex: Sistema de RH de uma empresa.
    - Temos a classe Funcionario e outras classes para cada atividade específica (Professor, terceirizado, técnico...)

Herança



- Com a criação dessa hierarquia, surgem dois novos conceitos:
  - Superclasse (classe mãe)
  - Subclasse (classe filha)
- O relacionamento criado entre uma superclasse e uma subclasse pode ser chamado de relacionamento “é um”.

Herança

- Dizemos que a subclasse estende da superclasse.
- Então, para criarmos uma herança em nosso sistema, utilizamos a palavra reservada **extends.**
- Ex: public class Professor **extends** Funcionario {

  }
- No exemplos acima, dizemos que o Professor "é um" Funcionario.
- Temos como superclasse o Funcionario e subclasse o Professor.

Herança


```java
public class Funcionario {
    private String nome;
    private int idade;
    private String data;
    private double salario;
    private String RG;
    // métodos getters e setters ...
}
```

```java
public class Professor extends Funcionario{

    public void ensinarDisciplina(String disc){
        System.out.println(getNome() +  " está ensinando " + disc);
    }

}
```

Herança

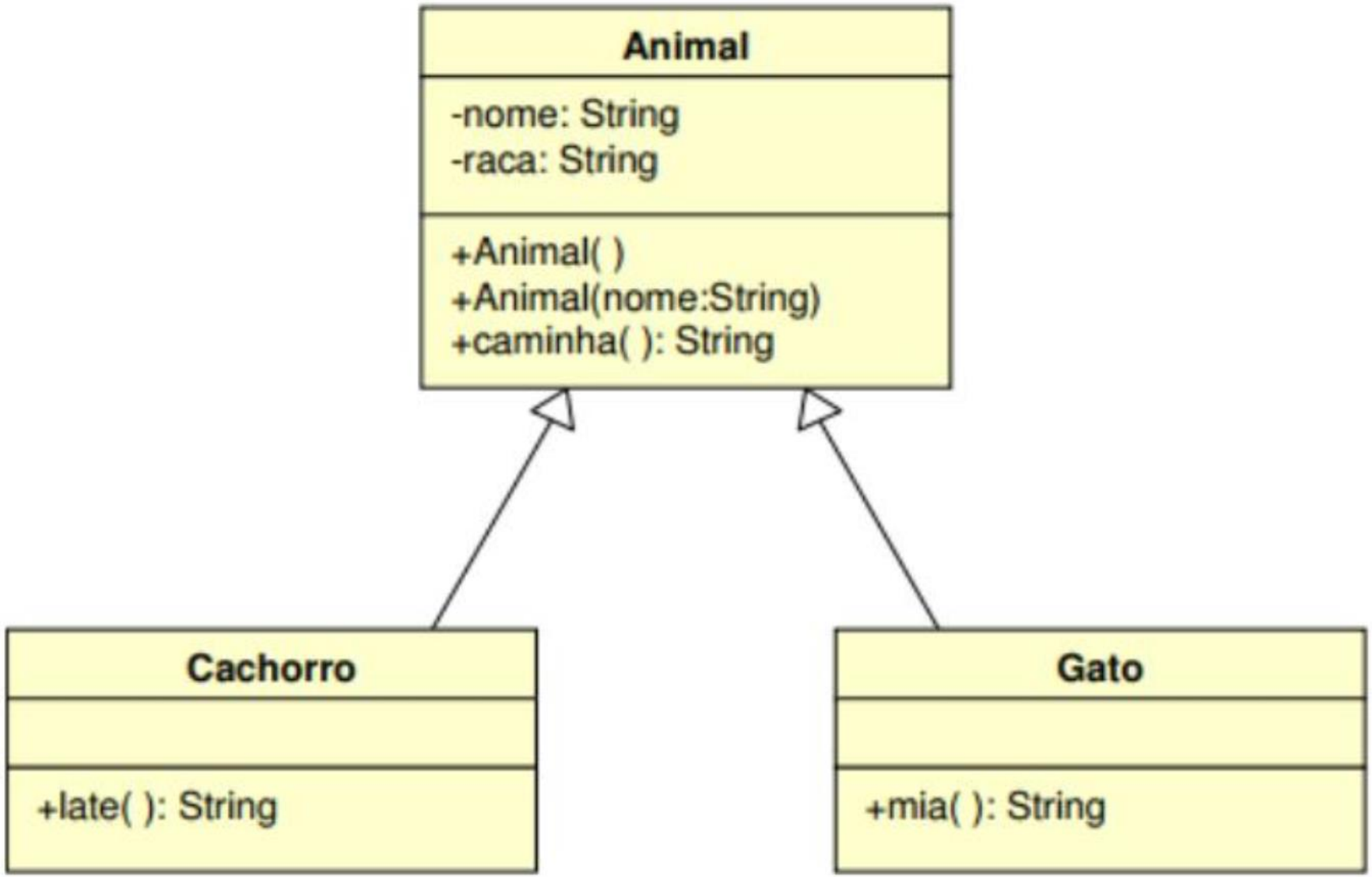
- Lembram do modificador de acesso **protected** ??
- É para ser utilizado juntamente com a herança.
- Ou seja, todos os atributos ou métodos com o modificador protected somente serão acessados por subclasses ou por outras classes pertencentes ao mesmo pacote.

Herança

- Se uma subclasse herda os comportamentos da superclasse, então toda subclasse vai fazer as mesmas coisas da superclasse ??

1. Podemos adicionar comportamentos exclusivos da subclasse;
2. Podemos sobrescrever um comportamento existente na superclasse.

Exemplo UML


Animal
-nome: String
-raca: String
+Animal( )
+Animal(nome:String)
+caminha( ): String
Cachorro
+late( ): String
Gato
+mia( ): String

Exercícios - HERANÇA

- Crie uma classe chamada Ingresso que possui um valor em reais e um método imprimeValor().

a. crie uma classe Normal, que herda de Ingresso e possui um método (nome a sua escolha) que imprime: "Ingresso Normal".

b. crie uma classe VIP, que herda de Ingresso e possui um valor adicional. Crie um método que retorne o valor do ingresso VIP (com o adicional incluído). OBS:: sobrescreva o método imprimeValor() da classe Ingresso.

c. crie uma classe CamaroteInferior (que possui a localização do ingresso e métodos para acessar (get e set) e imprimir esta localização) e uma classe CamaroteSuperior, que é mais cara (possui valor adicional). Esta última possui um método para retornar o valor do ingresso. Ambas as classes herdam a classe VIP.

Referências

- **Apostila Caelum:** https://www.caelum.com.br/apostila-java-orientacao-objetos/orientacao-a-objetos-basica
- H.M. Deitel, P.J. Deitel, **Java Como programar.**