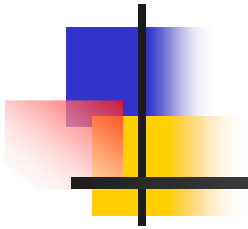


SIN143 Laboratório de Programação

Prof. João Batista Ribeiro

joao42lbatista@gmail.com

UFV

Universidade Federal de Viçosa

Aula de hoje

- Introdução ao(s) trabalhos(s) de SIN 143
- Revisão a Orientação a Objetos (OO)
- Java (micro introdução)

Trabalho(s) de SIN 143

Grupo de até 3 alunos → **Estilo maratona de programação**

- Engenharia
- Programação (POO)
- Banco de dados

- **Nome dos integrantes (e matricula) e tema para próxima aula**

Algumas ideias…

- Biblioteca, Loja de informática, Mercearia, Pizzaria, Gerenciamento Esportivo, Acompanhamento de pacientes, Clinica, Organização de eventos, Madeireira, Padaria, Auto Peças, Imobiliária, Lava Jato, Banco, Concessionária, Gerência de (Escola de idiomas, Cinema, Ordem de serviços etc) etc

Desenvolvimento de software

Abstração do problema

- Princípio básico do desenvolvimento de software
- Retirar do domínio do problema detalhes relevantes e representá-los na linguagem da solução

Paradigma (de programação)

- Forma de como atacar um problema
- A evolução das linguagens de programação influenciaram a forma como os problemas são atacados
- A tecnologia de cada época delimitou como os problemas eram atacados

Paradigma de programação

Paradigma procedural/estruturada:

- Fundamentada sobre estruturas de sequência, decisão e repetição
- A solução de cada pequena parte do problema é feita em procedimentos(ou funções) e a solução de todo problema consiste na invocação destes procedimentos
- Dividir para conquistar!
- Visa a reutilização de código
- Exemplos: Pascal e C

Paradigma de programação

Paradigma orientado a objetos:

- Ideal para o desenvolvimento de software complexo, porém traz complexidade desnecessária para projetos pequenos
- Todo o sistema é visualizado como um conjunto de células interconectadas, denominadas objetos
- Cada objeto possui uma tarefa específica e por meio da troca de mensagens entre os objetos é possível realizar uma tarefa computacional
- Reutilização de código e facilidade de manutenção
- Exemplos:C++ e Java

POO

Programação Orientação a Objetos (POO)

- Representar elementos do mundo real em forma de objetos

Presente nas fases de análise, levantamento de requisitos, construção do software etc

- Unified Modeling Language (UML)

POO

Objetos → ~~Tudo~~

- Aglutinação de estados e comportamentos
- Estado → Características/Propriedades → Atributos
- Comportamento → Funcionalidades/Funções/Operações → Métodos

Caneta

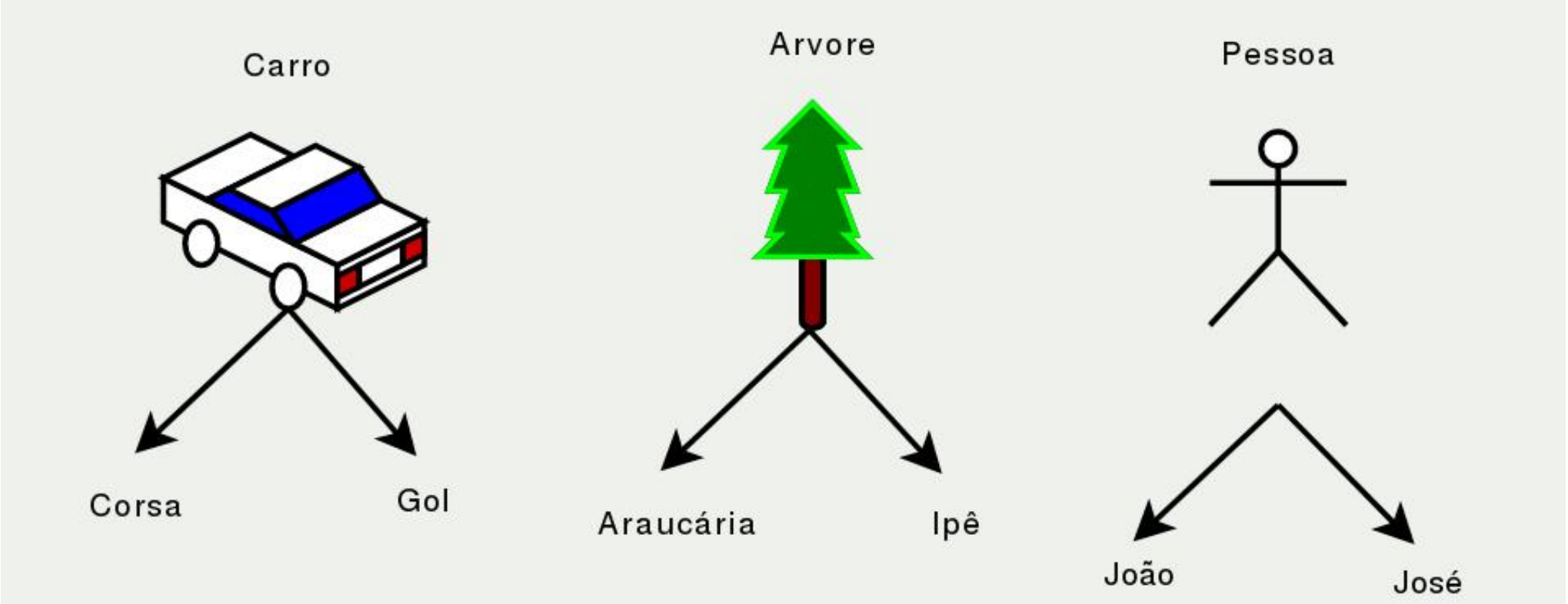
- Estado: Preta, para papel, com 80% de tinta
- Comportamento: Escreve, rebobina fita

Classes

- Atributos (public, private etc), this.Atributo
- Métodos (gets, sets, outroComportamento etc)
  - Construtor, sobrecarga de construtor, sobrecarga de método

Exemplos: carro (mustang, uno), pessoa (cliente, empregado)

Classe e objetos



POO e seus pilares

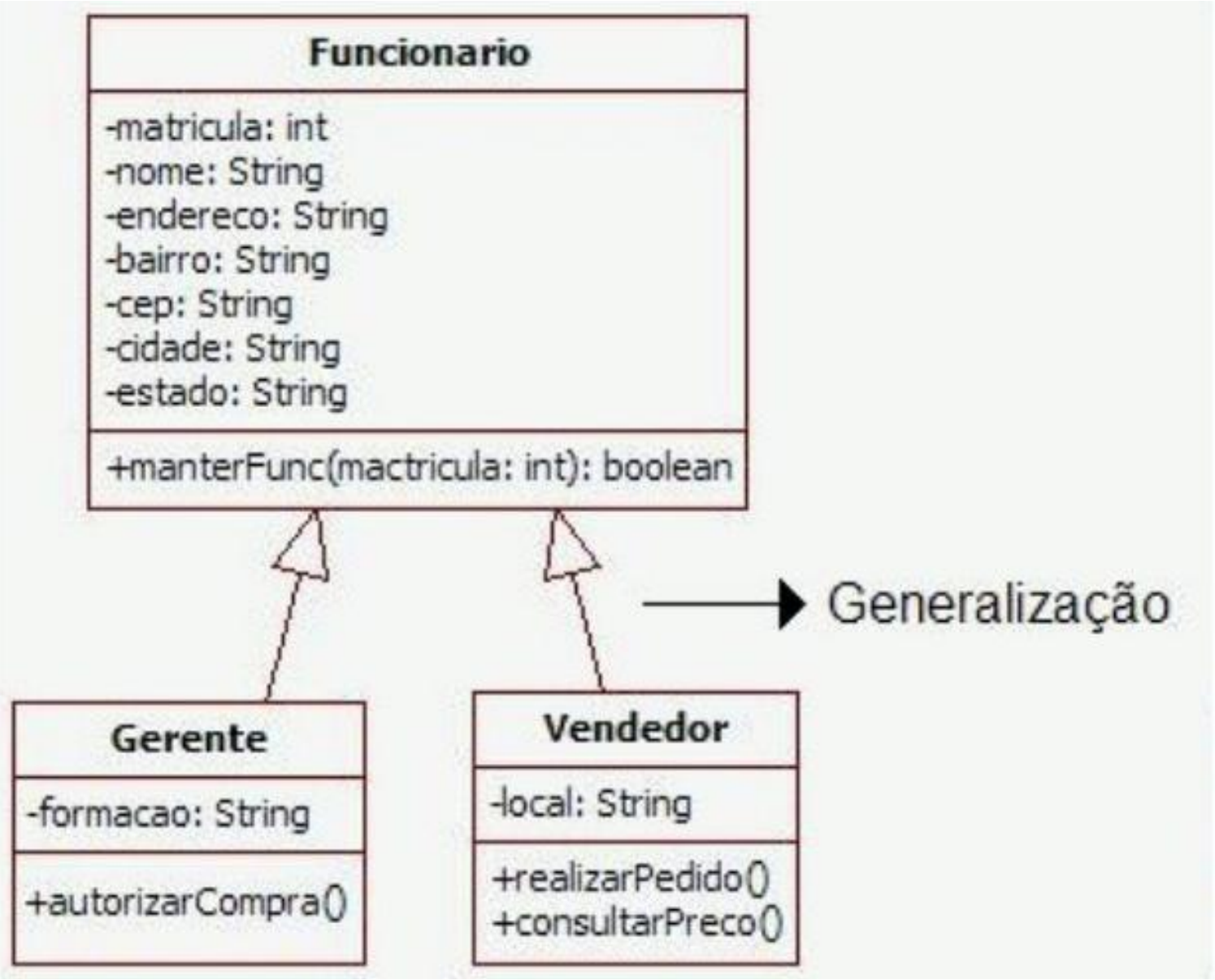
- Herança
- Polimorfismo
- Encapsulamento
- Abstração

POO e seus pilares

Herança

- Idealização de um objeto baseado em outro objeto
- Objeto “pai” são estendidos para um novo objeto “filho”. “Pai” também pode já ter sido estendido
- “Caneta pai” – Tem tampa e escreve
- “Caneta filho1” - Tampa azul, escreve, rebobina fita
- Estender métodos:
- “caneta filho1” escreve com uma cor diferente da “caneta pai”
- Reutilização de código

Herança


Funcionario
-matricula: int
-nome: String
-endereco: String
-bairro: String
-cep: String
-cidade: String
-estado: String
+manterFunc(mactricula: int): boolean
Generalização
Gerente
-formacao: String
+autorizarCompra()
Vendedor
-local: String
+realizarPedido()
+consultarPreco()

Herança

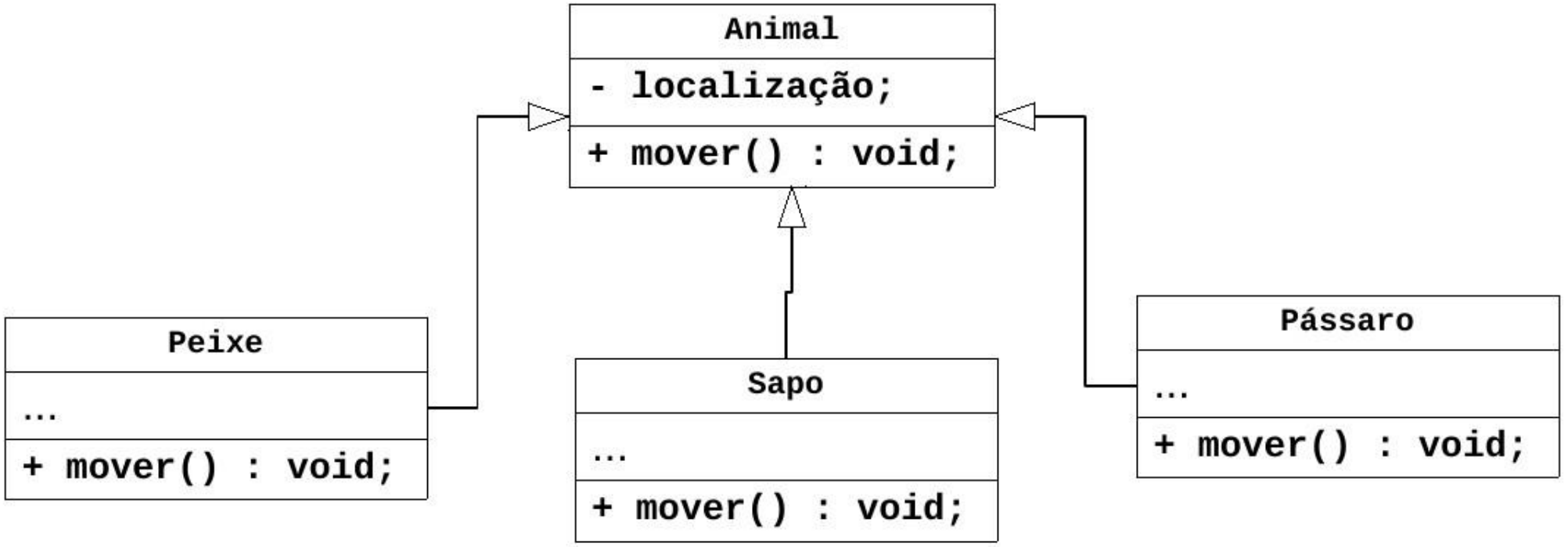

MembroDaComunidade
Empregado
Aluno
Graduados
Herança única
CorpoDocente
Funcionários
Herança única
Administrador
Professor
Herança única
ProfessorAdministrador
Herança múltipla

POO e seus pilares

Polimorfismo

- Capacidade de um objeto (que sejam compatível) passar por outro em um determinada circunstância
- Objeto herdado → trás características do seu “pai”, mas algumas podem precisar ser rescritas/recodificada
- Rescrita de um método herdado sem utiliza nenhum comportamento da classe pai

Polimorfismo



POO e seus pilares

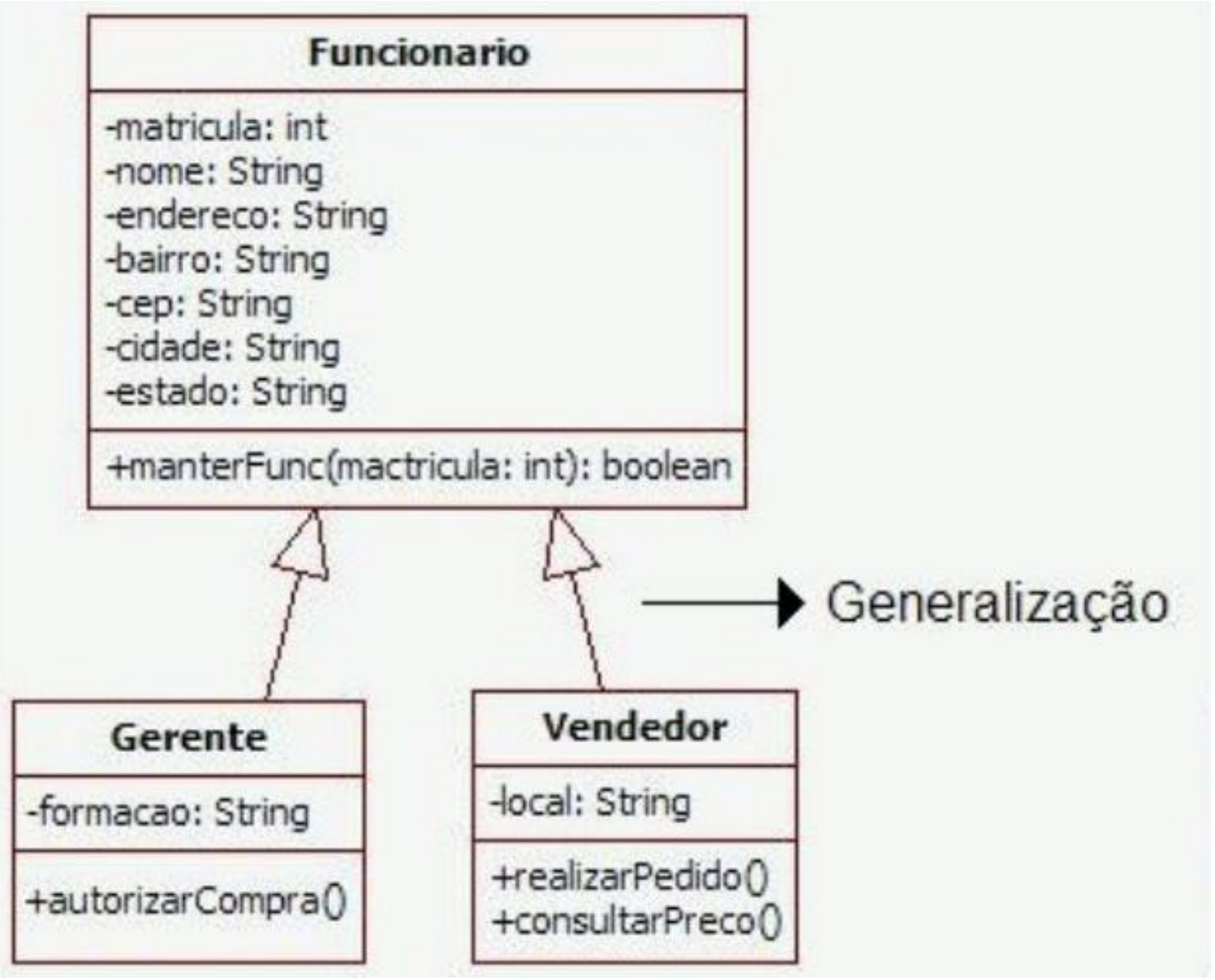
Encapsulamento

- Capacidade de esconder detalhes da implementação do objeto → expondo apenas o que deve ser acessado publicamente
- Atributos → private → sets/gets
- Métodos → public → outras funções

Adiciona segurança a aplicação

- Criar “caixa preta” em seus objetos

Encapsulamento + Herança

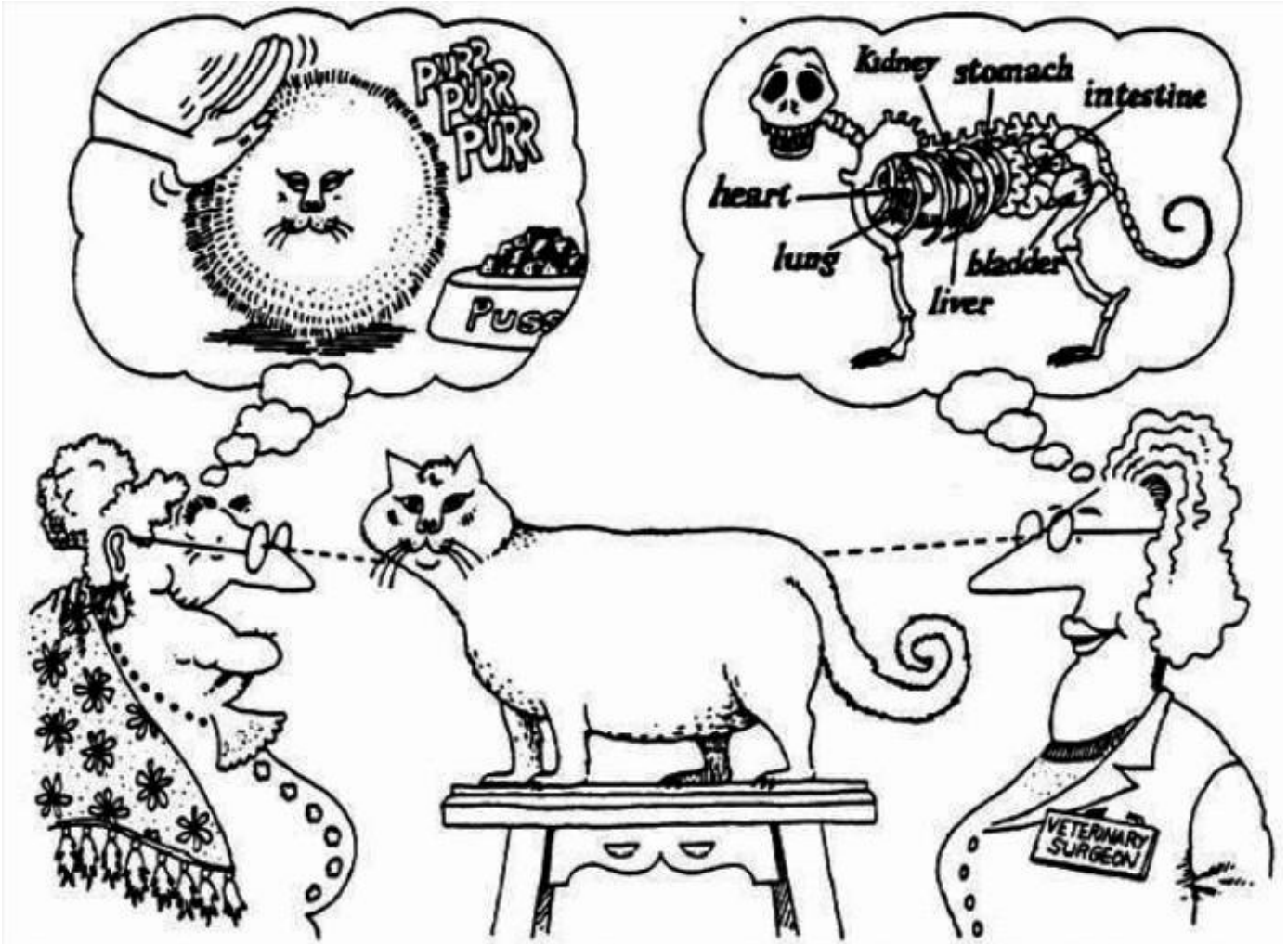


POO e seus pilares

Abstração

- Representar um objeto de forma uma abstrata, que seja obrigatoriamente herdado por outras classes
- Classe abstrata com atributos e métodos, mas a implementação precisa ser feita nas classes “filhas”
- Template
- Interface
  - Não pode ser criado objetos diretamente dela

Abstração



Abstraction focuses upon the essential characteristics of some object, relative to the perspective of the viewer.

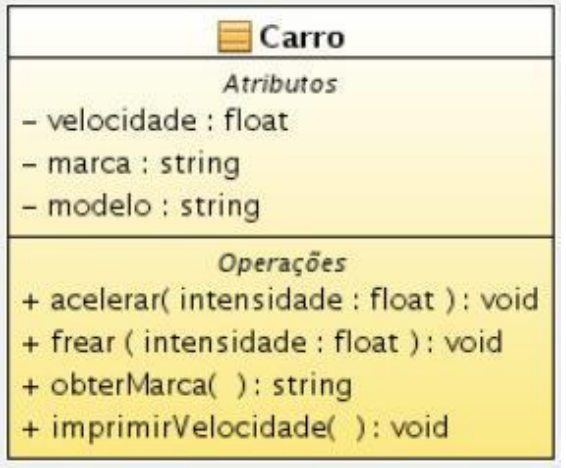
Java

- Neutro em relação à arquitetura
  - Bytecode → JVM (máquina virtual Java)
  - Várias bibliotecas
  - Java Runtime Environment (JRE)
    - JVM
  - Java Development Kit (JDK)
    - Compilador, JVM, código fonte, documentação das APIs
    - Distribuído gratuitamente pela Oracle para diversos sistemas operacionais e arquiteturas de máquina

Java

- Standard Edition – Java SE
  - Aplicações desktop
- Enterprise Edition – Java EE
  - aplicações empresariais
- Mobile Edition – Java ME
  - aplicações para dispositivos móveis
- IDE
  - Eclipse, Netbeans
  - Verificar qual está instalado no laboratório

Java exemplo de código




```
11 public class Carro{
12   // atributos
13   private double velocidade;
14   private String marca;
15   private String modelo;
16   // metodos
17   public void acelerar(double intensidade){
18     ...
19     }
20   public void frear(double intensidade){
21     ...
22     }
23   public String obterMarca(){
24       return marca;
25     }
26   public void imprimirVelocidade(){
27       System.out.println("Velocidade: " + velocidade);
28     }
29 }
```

Java exemplo de código

```
public class Conta {
    private double saldo;
    public Conta ( ) {
    }
    public Conta (double saldo) {
        this.saldo = saldo;
    }
    public void creditar (double valor) {
        saldo = saldo + valor;
    }
    public void debitar (double valor) {
        saldo = saldo - valor;
    }
    public double obterSaldo ( ) {
        return saldo;
    }
}
```

Construtor *default*

Construtor com um argumento

Os construtores normalmente inicializam os atributos da classe.

Obrigado pela atenção :)

Recomendação de leitura: https://www.caelum.com.br/apostila-java-orientacao-objetos/