PROGRAMAÇÃO ORIENTADA A OBJETOS

João Choma Neto

joao.choma@unicesumar.edu.br

Unicesumar – Maringá

TEMAS DE ESTUDO

- *Características de linguagens de programação, compilação e interpretação de linguagens, linguagens tipadas e não tipadas.*

TEMAS DE ESTUDO

- *Tipos, variáveis e atribuições.*
- *Objetos, classes e métodos. Parâmetros e valores de retorno. Tipos numéricos*

TEMAS DE ESTUDO

- *Construção de objetos, métodos de acesso e modificadores. Referências a objetos.*
- *Testes unitários.*
- *Encapsulamento de atributos de objetos*

TEMAS DE ESTUDO

- *Associações simples entre classes usando atributos de instância.*
- *Uso de arrays na linguagem Java.*
- *Uso de ArrayList comparado ao uso de arrays.*

TEMAS DE ESTUDO

- *Herança*
- *Interfaces*
- *Classes abstratas*
- *Polimorfismo*
- *Uso de frameworks de testes unitários*
- *Uso de frameworks de interface e dados*

BIBLIOGRAFIA BÁSICA

- *HORSTMANN, Cay S. Big Java / 2006 Porto Alegre: Bookman, 2006.*
- *SCHILDT, Herbert; SILVA, Aldir Coelho Corrêa da. Java para iniciantes - 5. ed. / 2013 Porto Alegre: Bookman, 2013.*
- *DEITEL, Harvey M; DEITEL, Paul J; DEITEL, Abbey. Android : como programar - 2 / 2015 Porto Alegre: Bookman, 2015.*

AVALIAÇÕES

- 1º Bimestre
- 1,0 - Atividade de Estudo Programada.
- 1,0 - Prova Integrada.
- 8,0 - Avaliação prática, sendo:
  - 3,0 - Atividades em sala
  - 5,0 - Prova Prática

- 2º Bimestre
- 1,0 - Atividade de Estudo Programada.
- 1,0 - Prova Integrada.
- 8,0 - Avaliação prática, sendo:
  - 3,0 - Atividades em sala
  - 5,0 - Prova Prática

PARADIGMA DE PROGRAMAÇÃO

- Um paradigma pode ser entendido como a **forma** com a qual se decide **resolver** determinado **problema** por meio da programação

PARADIGMA DE PROGRAMAÇÃO

- Programação imperativa
- Paradigma de programação mais comum
- Utiliza instruções que modificam o estado de variáveis para controlar o fluxo de execução do programa

- Programação imperativa

```c
#include <stdio.h>

int main() {
   int x = 5;

   if (x > 0) {
      printf("x é positivo");
   } else {
      printf("x é não-positivo");
   }

   return 0;
}
```

Linguagem de programação

- Programação funcional
- Paradigma de programação se concentra no uso de funções matemáticas
- Utiliza funções para transformar dados

- Programação funcional

```
; Definir uma função para calcular o fatorial de um número
(define (fatorial n)
  (if (<= n 1)
      1
      (* n (fatorial (- n 1)))))

; Definir uma função para calcular a soma dos elementos de uma lista
(define (soma-lista lst)
  (cond
    ((null? lst) 0)
    (else (+ (car lst) (soma-lista (cdr lst))))))

; Definir uma função para calcular a sequência de Fibonacci
(define (fibonacci n)
  (cond
    ((= n 0) 0)
    ((= n 1) 1)
    (else (+ (fibonacci (- n 1)) (fibonacci (- n 2))))))

; Exemplo de uso das funções
(displayln (fatorial 5)) ; Imprime "120"
(displayln (soma-lista '(1 2 3 4 5))) ; Imprime "15"
(displayln (fibonacci 10)) ; Imprime "55"
```

Linguagem de programação

- Programação baseada em lógica
- Paradigma de programação usa a lógica matemática para resolver problemas de programação
- Baseado em regras e fatos lógicos

- Programação funcional

```
ir fatos para relacionar algumas cores com alguma
e(joao, laranja).
e(maria, banana).
e(pedro, abacaxi).
e(joana, abacaxi).
e(paulo, laranja).
e(ana, uva).

ir regras para relacionar pessoas com suas frutas
referida(Pessoa, Fruta) :-
ta_de(Pessoa, Fruta),
(gosta_de(outra_pessoa, Fruta)),
(gosta_de(Pessoa, outra_fruta)).

ir uma consulta para encontrar a fruta preferida
a_preferida(joao, Fruta).
```

Programação Concorrente

- Programação concorrente
- Paradigma de programação se concentra na execução simultânea de várias tarefas ou processos

- Programação concorrente

```
port java.util.concurrent.ExecutorService;
port java.util.concurrent.Executors;

blic class ExemploConcorrente {

  public static void main(String[] args) {
      // Criar um pool de threads com duas threads
      ExecutorService executor = Executors.newFixedThre

      // Executar duas tarefas concorrentemente
      executor.submit(new TarefaConcorrente("Tarefa 1")
      executor.submit(new TarefaConcorrente("Tarefa 2")

      // Encerrar o pool de threads quando as tarefas e
      executor.shutdown();
  }
```

- Programação concorrente

```
class TarefaConcorrente implements Runnable {

    private String nome;

    public TarefaConcorrente(String nome) {
        this.nome = nome;
    }

    public void run() {
        System.out.println("Iniciando " + nome);
        try {
            // Dormir por um período aleatório de tempo para
            Thread.sleep((long) (Math.random() * 5000));
        } catch (InterruptedException e) {
            System.out.println("Interrupção durante " + nome
        }
        System.out.println("Concluindo " + nome);
    }

}
```

Linguagem de programação

- Programação orientada a objetos (POO)
- Paradigma de programação enfatiza a organização do código em classes e objetos
- A tarefas são realizadas pela interação dos objetos
- Abstração do mundo real em objetos

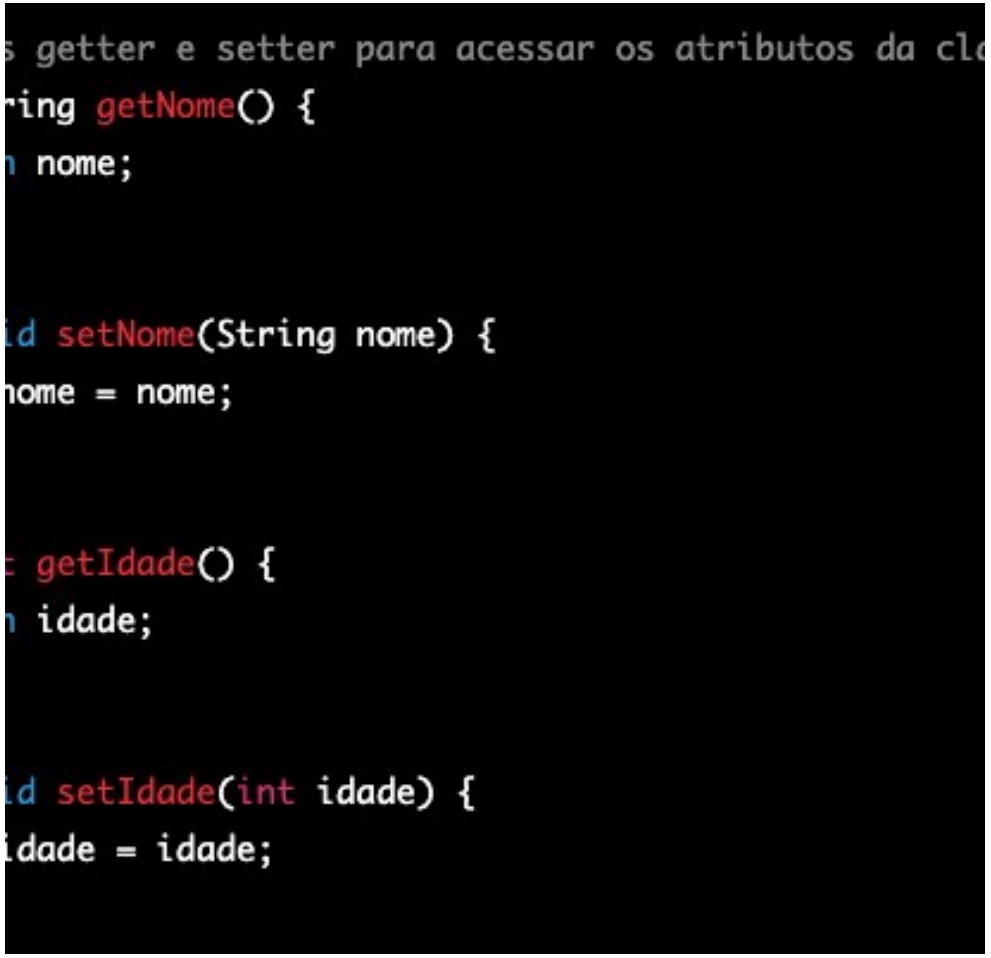
- Programação orientada a objetos

```
ic class Pessoa {

// Atributos da classe Pessoa
private String nome;
private int idade;

// Construtor da classe Pessoa
public Pessoa(String nome, int idad
    this.nome = nome;
    this.idade = idade;
}
```

- Programação orientada a objetos



- Programação orientada a objetos

```
in para testar a classe Pessoa
c void main(String[] args) {
 uma nova instância da classe Pessoa
essoa1 = new Pessoa("João", 30);

r o método saudar da instância pessoa1
saudar();

ar o nome e idade da instância pessoa1 usando
setNome("Maria");
setIdade(25);

r o método saudar da instância pessoa1 novamer
saudar();
```

Programação Orientada a Objetos

POO

- O paradigma da POO (Programação Orientada a Objetos) é um modelo de análise, projeto e programação baseado na aproximação entre o mundo real e o mundo virtual

- Criação e interação entre objetos

POO

- A primeira linguagem de programação com paradigma de orientação a objetos foi criada em 1970

- Criado por Alan Kay

POO

- Smalltalk

- Primeira linguagem a usar conceitos de classes, objetos, atributos e métodos.

POO – Abstração do mundo real

- Modelar o comportamento de entidades do mundo real como objetos em software
- Os objetos são entidades que possuem valores/pesos/dados e comportamentos

POO – Vantagens

- Confiável - qualquer intervenção que seja necessária não afetará outros pontos do sistema
- Ajustável – a herança garante que outras partes que utilizam uma classe sejam beneficiadas
- Extensível – reutilização de código

POO – Vantagens

- Reutilizável – um mesmo objeto pode ser utilizado em diferentes sistemas (objeto cliente)
- Natural – abstração do mundo real

POO - Base

- Encapsulamento
- O encapsulamento é a capacidade que determinado método ou atributo de um objeto tem de se manter invisível
- É aquele famoso pensamento de saber o que faz, mas não saber como se faz

POO - Base

- Herança
- Trata-se de uma relação de receber algo pré-existente
- Classe Mãe doa seu funcionamento a classe filha

POO - Base

- Polimorfismo
- Um mesmo método pode ser utilizado em diferentes objetos, de diferentes classes.

POO - Base

- Abstração
- A ideia principal é representar um objeto de forma abstrata
- Uma classe abstrata contem atributos e métodos
- Sua implementação é de responsabilidade da classe que herda a classe abstrata

Linguagem Trabalhada Java

Linguagem Java

- A linguagem Java carrega heranças de muitas linguagens
    - Algol
    - CPL
    - B
    - C
    - C++

Origem da linguagem Java

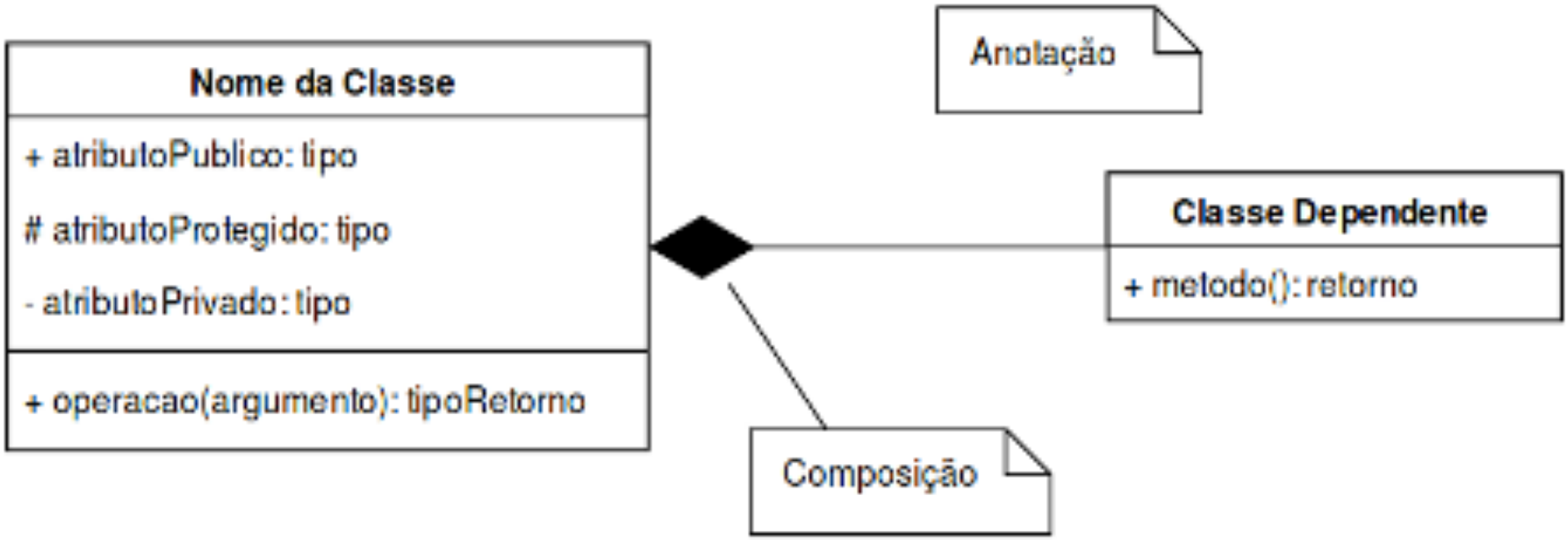
- 1991
- Parecida com C++
- Fortemente tipada
- Portável – Roda em uma máquina virtual

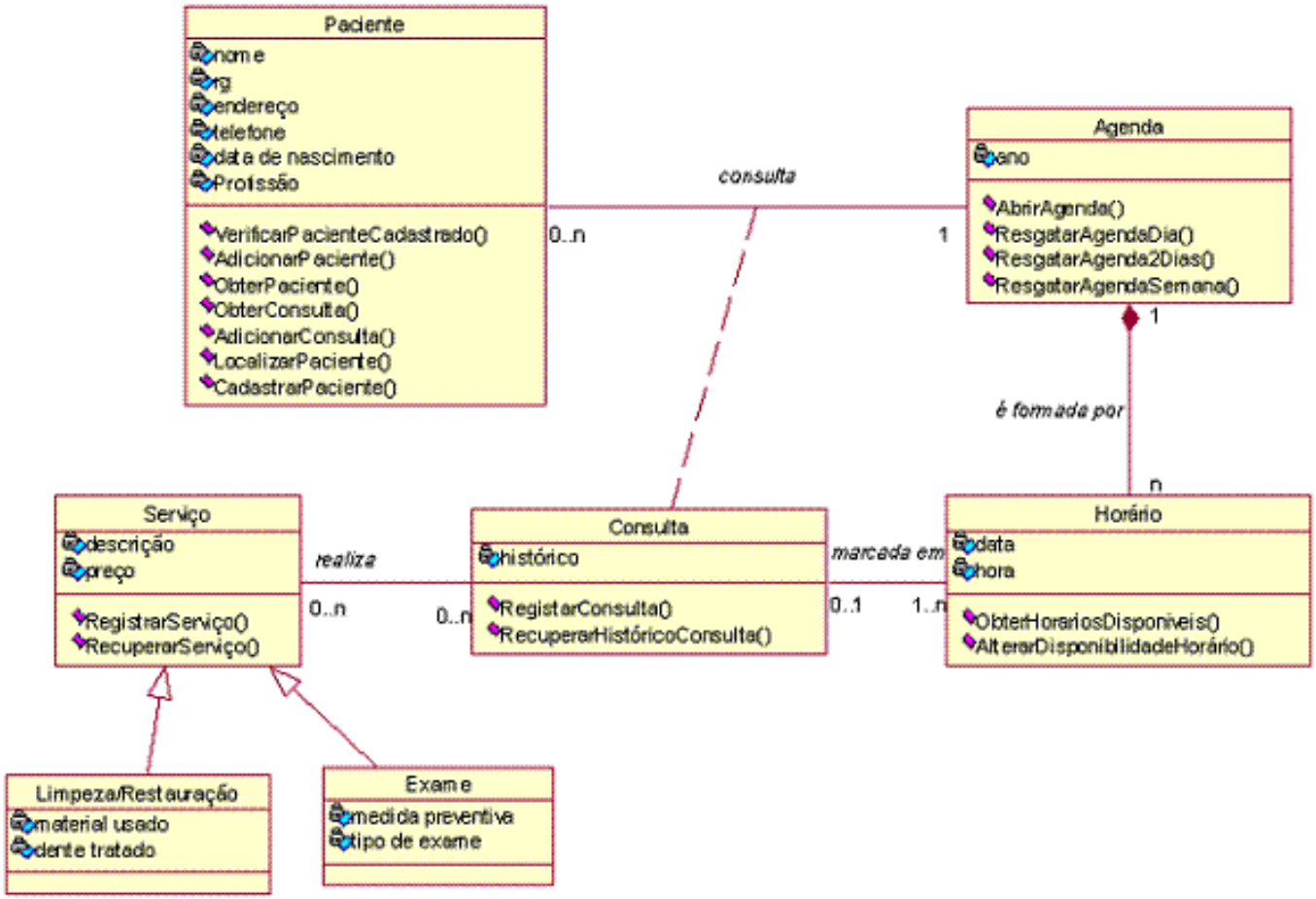
- Recebeu o nome de Java em 1995

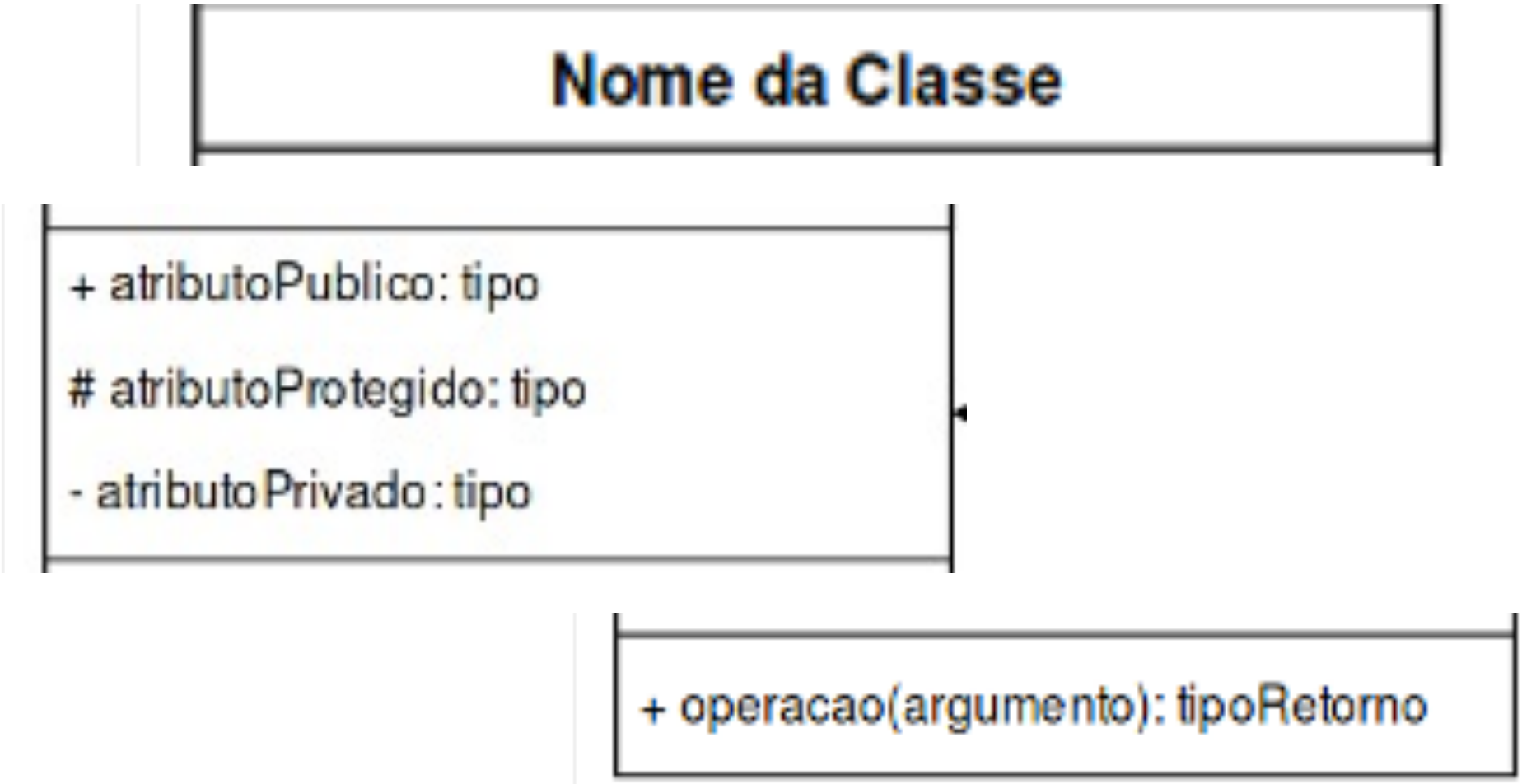
RELEMBRAR

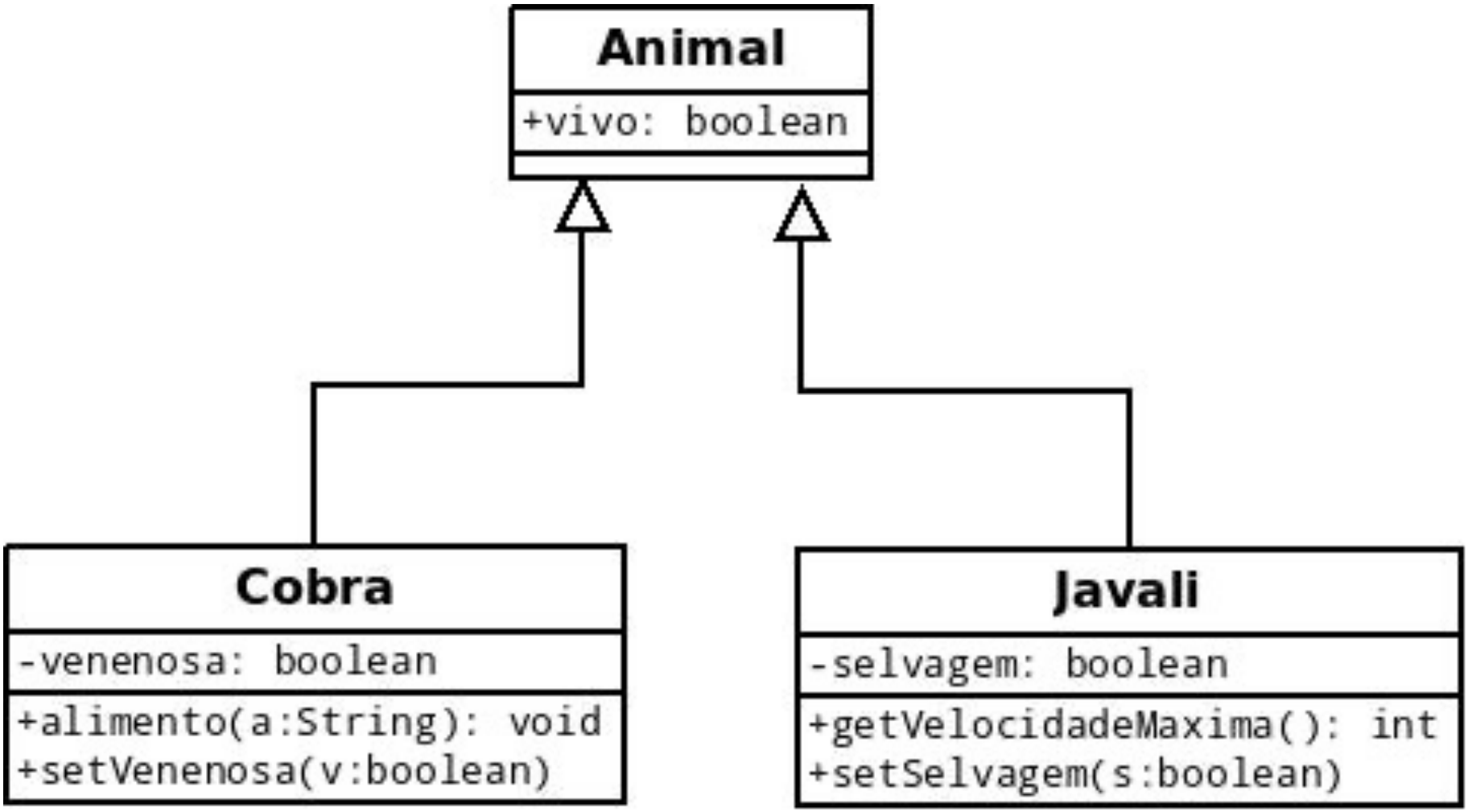
POO

- Na programação orientada a objetos (POO), uma classe é um modelo que define as propriedades (atributos) e comportamentos (métodos) que os objetos desse tipo podem ter.
- Uma classe é uma representação abstrata de um objeto do mundo real, que encapsula suas características e comportamentos.


Nome da Classe
+ atributoPublico: tipo
# atributoProtegido: tipo
- atributoPrivado: tipo
+ operacao(argumento): tipoRetorno
Anotação
Classe Dependente
+ metodo(): retorno
Composição


Paciente
nome
rg
endereço
telefone
data de nascimento
Profissão
VerificarPacienteCadastrado()
AdicionarPaciente()
ObterPaciente()
ObterConsulta()
AdicionarConsulta()
LocalizarPaciente()
CadastrarPaciente()
consulta
0..n
1
Agenda
ano
AbrirAgenda()
ResgatarAgendaDia()
ResgatarAgenda2Dias()
ResgatarAgendaSemana()
1
é formada por
n
Serviço
descrição
preço
RegistrarServiço()
RecuperarServiço()
realiza
0..n
0..n
Consulta
histórico
RegistarConsulta()
RecuperarHistóricoConsulta()
marcada em
0..1
1..n
Horário
data
hora
ObterHorariosDisponiveis()
AlterarDisponibilidadeHorário()
Limpeza/Restauração
material usado
dente tratado
Exame
medida preventiva
tipo de exame


Nome da Classe
+ atributoPublico: tipo
# atributoProtegido: tipo
- atributoPrivado: tipo
+ operacao(argumento): tipoRetorno


Animal
+vivo: boolean
Cobra
-venenosa: boolean
+alimento(a:String): void
+setVenenosa(v:boolean)
Javali
-selvagem: boolean
+getVelocidadeMaxima(): int
+setSelvagem(s:boolean)