

Paradigmas

Módulo Básico

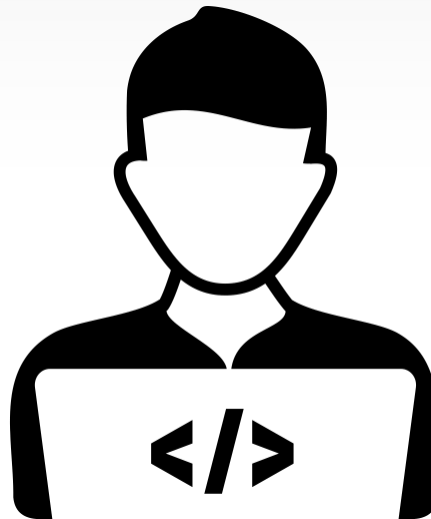




PARADIGMAS DE PROGRAMAÇÃO

Introdução

- ***Apresentação***
 - *O estudo dos **Paradigmas** de **Programação** ao profissional de tecnologia **compreender** os principais conceitos associados às **linguagens de programação***



- **Ementa**
 - *Fundamentos da construção de algoritmos, linguagens de programação e programas*
 - *Compiladores, tradutores e interpretadores*
 - *Evolução das linguagens de programação*
 - *Processo de desenvolvimento de algoritmos e programas*
 - *Paradigmas de programação: estruturada, funcional, lógica e orientada a objetos*

Paradigmas

- **Paradigmas de Programação**
 - **Modelo** para **estruturar** e **representar problemas**, cuja **solução** deseja-se obter **por meio** de um **programa**, construído a partir de uma **linguagem de programação**
 - **Tipos de Paradigmas:**
 - **Imperativo**
 - **Lógico**
 - **Funcional**
 - **Orientado a eventos**
 - **Orientado a objetos**
 - **Etc.**

Paradigmas

- **Paradigmas de Programação**
 - *Algumas linguagens de programação permite programar em mais de um paradigma*
 - **Exemplo:**
 - *Desenvolvimento baseado no **paradigma imperativo** e, também, **orientado a objeto***

Depende exclusivamente do conhecimento do desenvolvedor a utilização de recursos mais adequados para a solução de cada um dos problemas

Paradigmas

- ***Por que estudar Paradigmas?***

- ***Benefícios:***

- ***Capacidade intelectual influenciada pelo poder expressivo das linguagens nas quais nos comunicamos, ou seja, quanto mais linguagens conhecermos, maior será a nossa capacidade intelectual***
 - ***Isso vale para as línguas nas quais nos comunicamos, por meio da linguagem natural (Português, Inglês, Italiano, Francês, etc.) e, também, por meio das diferentes linguagens de programação por meio das quais nós, desenvolvedores de software, podemos nos expressar para solucionar problemas***

Paradigmas

- *Por que estudar Paradigmas?*

- *Benefícios:*

- *Limitar-se à linguagem natural **limita** a capacidade de expressar os pensamentos em termos de profundidade e abstração*
 - *A **abstração** nos permite, literalmente, **abstrair detalhes**, para que possamos nos **focar** na **solução do problema**, de forma **mais genérica***
 - *À medida que vamos **quebrando** o **problema** em partes, podemos **aprofundar** os **detalhes** necessários para sua **implementação***
 - *A **abstração** também **compreende** a **definição** de **classes** e **métodos** (no chamado **POO** - **Paradigma Orientado a Objeto**) e no uso de **diferentes estruturas de dados***

Paradigmas

- *Por que estudar Paradigmas?*

- *Benefícios:*

- *O conhecimento de uma variedade mais ampla de recursos de linguagens de programação **reduz** as limitações no desenvolvimento de software, ou seja, podemos encontrar diferentes formas de resolver problemas, mesmo que a linguagem de programação **não** ofereça os recursos necessários, de forma nativa*
 - *Exemplo: vamos supor que precisemos utilizar uma estrutura de dados do tipo **pilha**. Entretanto, a linguagem de programação que estamos utilizando **não** suporta ponteiros. Podemos, então, criar uma **pilha** utilizando uma estrutura de dados mais simples, um vetor (ou matriz unidimensional)*

Paradigmas

- ***Por que estudar Paradigmas?***

- ***Benefícios:***

- ***Maior conhecimento para a escolha de linguagens apropriadas:***
 - Quanto mais **linguagens de programação** conhecemos, **maior** será a nossa **capacidade** de definir qual a **linguagem** mais **adequada** de acordo com o **tipo** de **problema** que **precisamos resolver**
 - ***Capacidade aumentada para aprender novas linguagens:***
 - Quanto mais **linguagens de programação** conhecemos, maior será o nosso **potencial** para **aprendermos** outras **linguagens de programação**, pois nosso cérebro já estará preparado para buscar as **características** mais **importantes** e **recursos diferenciados** de **cada linguagem**

Paradigmas

- ***Domínios de Programação***

- ***Aplicações Científicas:***

- *Os primeiros computadores digitais (década de 40) foram inventados para aplicações científicas: estruturas de dados simples com um grande número de operações aritméticas com ponto-flutuante*
 - *Entre as linguagens de programação para o desenvolvimento de aplicações científicas citam-se o Algol e o Fortran*



Paradigmas

- **Domínios de Programação**

- **Aplicações Comerciais:**

- O uso de **computadores** para **aplicações comerciais** iniciou-se na **década de 50**
 - As **linguagens de programação** destinadas ao **desenvolvimento** deste **tipo de aplicação** possuem **facilidades** para produzir **relatórios** e **armazenar dados**
 - Exemplos deste tipo de **linguagem de programação** são a **linguagem COBOL** (Common Business Oriented Language) e o **Clipper** (Clipper/dBase)

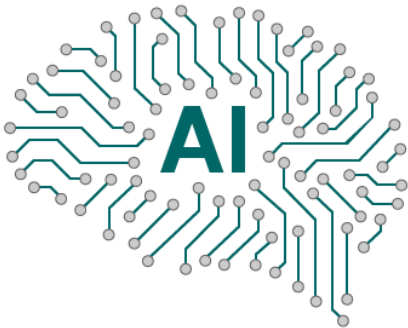


Paradigmas

- **Domínios de Programação**

- **Aplicações de IA (Inteligência Artificial):**

- São **aplicações computacionais** caracterizadas pelo uso de **computações simbólicas** ao invés de **numéricas**
 - Uma **lista**, por exemplo, pode **armazenar conhecimento** sobre um **determinado problema**, ao invés de **dados**
 - **Sistemas de IA** processam **conhecimento**, **não dados**
 - Entre as **linguagens** que podem ser **aplicadas** no **contexto da IA** destacam-se **LISP** e **PROLOG** (PROgramming in LOGic)



Paradigmas

- ***Domínios de Programação***
 - ***Programação de Sistemas:***
 - ***Compreendem as linguagens de programação para o desenvolvimento de **Sistemas Operacionais** (software básico)***
 - ***Os Sistemas Operacionais precisam ter **execução rápida** e recursos de **baixo nível** para permitir a **interface** com os dispositivos externos***
 - ***Entre as linguagens de programação com este propósito citam-se: **PL/S, BLISS, Extended ALGOL** e **C*****
 - ***O Sistema Operacional Unix (precursor do Linux) foi desenvolvido em **Linguagem C*****

Paradigmas

- **Domínios de Programação**

- **Linguagens de Scripting:**

- Caracterizada por uma **lista de comandos** (**script**) em um **arquivo** para serem executados

- Exemplo: **arquivos BATCH** (arquivos com a **extensão .BAT**). Era muito comum **criarmos arquivos .BAT** que continham **diversos comandos**, tais como **comandos de configuração do sistema**, que **deviam ser executado** ao **ligarmos o computador** (**AUTOEXEC.BAT** - um **arquivo autoexecutável**)



- Existem **outros tipos de linguagens de script**, tais como: **Action Script** (executada no **Adobe Flash**), **Open Script** (executada no **Multimedia ToolBook Instructor**) e a linguagem **Python**, entre outras

Paradigmas

- ***Domínios de Programação***
 - ***Linguagens de programação para **propósitos especiais**, tais como linguagens para a geração de **relatórios**, criação de **jogos** e **simulação** de sistemas***
 - ***Exemplos:***
 - ***A linguagem RPG (Report Program Generator), para geração de relatórios comerciais e a linguagem de simulação de sistemas GPSS (General Purpose Simulation System)***

Paradigmas

- ***Critérios de Avaliação (Linguagem de Programação)***
 - ***Legibilidade***
 - ***Facilidade com que os programas podem ser lidos e entendidos***
 - ***Este entendimento é importante para a manutenção dos programas***
 - ***Antes de 1970 o desenvolvimento de software era baseado na escrita de código, ou seja, o mais importante era, efetivamente, escrever os programas***



Paradigmas

- ***Critérios de Avaliação (Linguagem de Programação)***
 - ***Legibilidade***
 - Na **década de 70** a **codificação** passou para o **segundo plano**, dando lugar à **manutenção** (a **facilidade** de **manutenção** é determinada pela **legibilidade** dos programas)
 - A **legibilidade** pode ser **avaliada** por meio das **seguintes características**:
 - (1) **simplicidade global**
 - (2) **ortogonalidade**
 - (3) **instruções de controle**
 - (4) **tipos de dados e estruturas**
 - (5) **sintaxe**

Paradigmas

- ***Critérios de Avaliação (Linguagem de Programação)***
 - ***Legibilidade: Simplicidade Global***
 - *Uma linguagem com um grande número de componentes básicos é mais difícil de ser aprendida do que uma com poucos desses componentes*
 - *Geralmente, os desenvolvedores de software que precisam usar uma linguagem grande tendem a aprender um subconjunto dela e ignorar seus outros recursos*
 - *Este é um exemplo do que acontece com linguagens de programação mais complexas, como é o caso de Java*

Paradigmas

- ***Critérios de Avaliação (Linguagem de Programação)***
 - ***Legibilidade: Simplicidade Global***
 - ***Problemas*** de ***legibilidade*** ocorrem sempre que o autor do programa tenha ***aprendido*** um subconjunto ***diferente*** daquele com o qual o leitor está ***familiarizado***
 - Programar é uma ***atividade criativa***, ou seja, ***não*** é algo que pode ser, a princípio, ***automatizado***, pois cada um de nós pode ***criar programas diferentes***, com ***recursos diferentes***, para ***resolver*** um mesmo ***problema***
 - A ***multiplicidade*** de ***recursos*** e a ***sobrecarga*** são ***características*** que ***decrementam*** a ***legibilidade***



Paradigmas

- ***Critérios de Avaliação (Linguagem de Programação)***
 - ***Legibilidade: Ortogonalidade***
 - *Significa que um conjunto relativamente pequeno de construções primitivas pode ser combinado em um número relativamente pequeno de maneiras para construir as estruturas de controle e de dados da linguagem*
 - *Os ponteiros devem ser capazes de apontar para qualquer tipo de variável ou estrutura de dados para que uma linguagem de programação seja ortogonal*

Paradigmas

- ***Critérios de Avaliação (Linguagem de Programação)***
 - ***Legibilidade: Instruções de Controle***
 - As ***instruções*** de ***controle*** que ***envolvem desvios*** (os ***chamados “gotos”***) ***dificultam a legibilidade*** de um programa
 - Para ***aumentar a legibilidade*** deve-se utilizar a ***programação estruturada*** e ***eliminar comandos de desvio (goto)***
 - Isso pode ser feito utilizando-se ***sub-rotinas (procedimentos e funções)*** e, em ***paradigmas*** como o da ***Orientação a Objeto***, por meio de ***métodos***



Paradigmas

- ***Critérios de Avaliação (Linguagem de Programação)***
 - ***Legibilidade: Tipos de Dados e Estruturas***
 - A ***presença de facilidades adequadas*** para definir ***tipos de dados e estruturas de dados*** em uma ***linguagem de programação*** é outro ***auxílio significativo*** para a ***legibilidade***
 - ***Algumas linguagens de programação mais modernas possuem estruturas de dados já implementadas, para que os desenvolvedores possam utilizar pilhas, filas, listas e outras estruturas***
 - ***Em algumas linguagens é preciso construir métodos para implementar estas estruturas, geralmente utilizando-se ponteiros (pointers – alocação dinâmica de memória)***

Paradigmas

- ***Critérios de Avaliação (Linguagem de Programação)***
 - ***Legibilidade: Considerações sobre a Sintaxe***
 - ***Restringir os identificadores a tamanhos muito pequenos (na linguagem FORTRAN 77, os identificadores podiam ter apenas 6 caracteres no máximo)***
 - ***Palavras especiais ou reservadas: a legibilidade de um programa é fortemente influenciada pelas formas das palavras especiais, tais como o método para formar instruções compostas ou grupos de instruções (begin – end, abrir e fechar chaves {})***
 - ***Utilizar os mesmos símbolos não é adequado (o mais correto seria utilizar end for, end if, end while, ao invés de usar somente fechar chaves)***

Paradigmas

- ***Critérios de Avaliação (Linguagem de Programação)***
 - ***Capacidade de Escrita (**Writability**)***
 - É a medida de **quão facilmente** uma **linguagem** pode ser utilizada para **criar programas** para um **domínio** de **problema** escolhido
 - A **maioria** das **características** da **linguagem** que **afeta** a **legibilidade** também **afeta** a **capacidade de escrita** (**escrever** um **programa** exige uma **releitura** frequente da parte que já foi **escrita** pelo **programador**) e é **essencial** para a **manutenção** de **sistemas**



Paradigmas

- ***Critérios de Avaliação (Linguagem de Programação)***
 - ***Custo***
 - O **custo final** de uma **linguagem de programação** é uma **função** de **muitas de suas características**
 - ***Custo para treinar desenvolvedores, compreendendo as características ligadas à simplicidade e à ortogonalidade***
 - ***Custo para escrever (writability) programas na linguagem***
 - ***Custo para compilar programas na linguagem***
 - ***Custo para executar programas***



Paradigmas

- ***Critérios de Avaliação (Linguagem de Programação)***
 - ***Custo***
 - O **custo final** de uma **linguagem de programação** é uma **função** de **muitas de suas características**
 - ***Custo do sistema de implementação da linguagem***
 - ***Custo da má confiabilidade***
 - ***Custo da manutenção de programas***





LINGUAGENS DE PROGRAMAÇÃO E ALGORITMOS

- ***Linguagens de Programação e Algoritmos***
 - ***Organização do Computador***

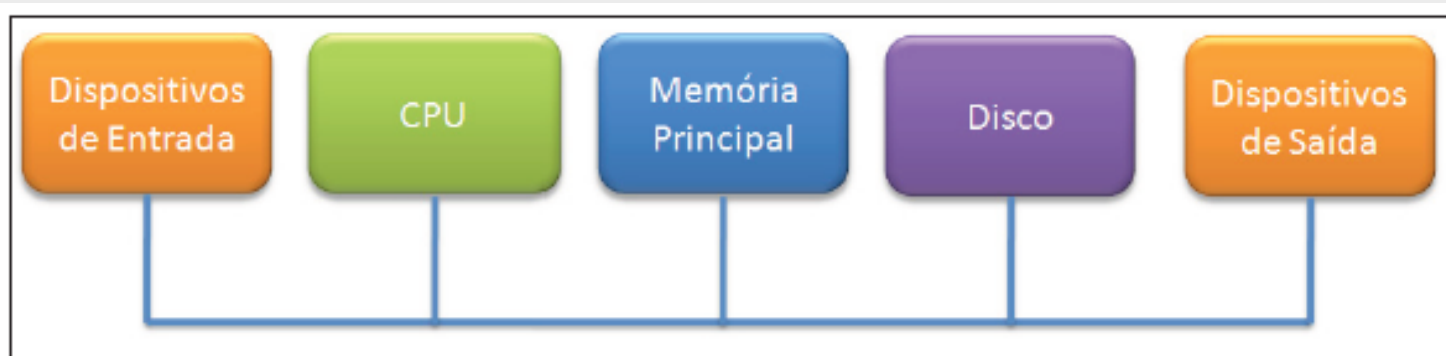


Figura 1 – Organização do Computador

- ***Linguagens de Programação e Algoritmos***
 - ***Linguagens de Programação***
 - ***Compiladas***

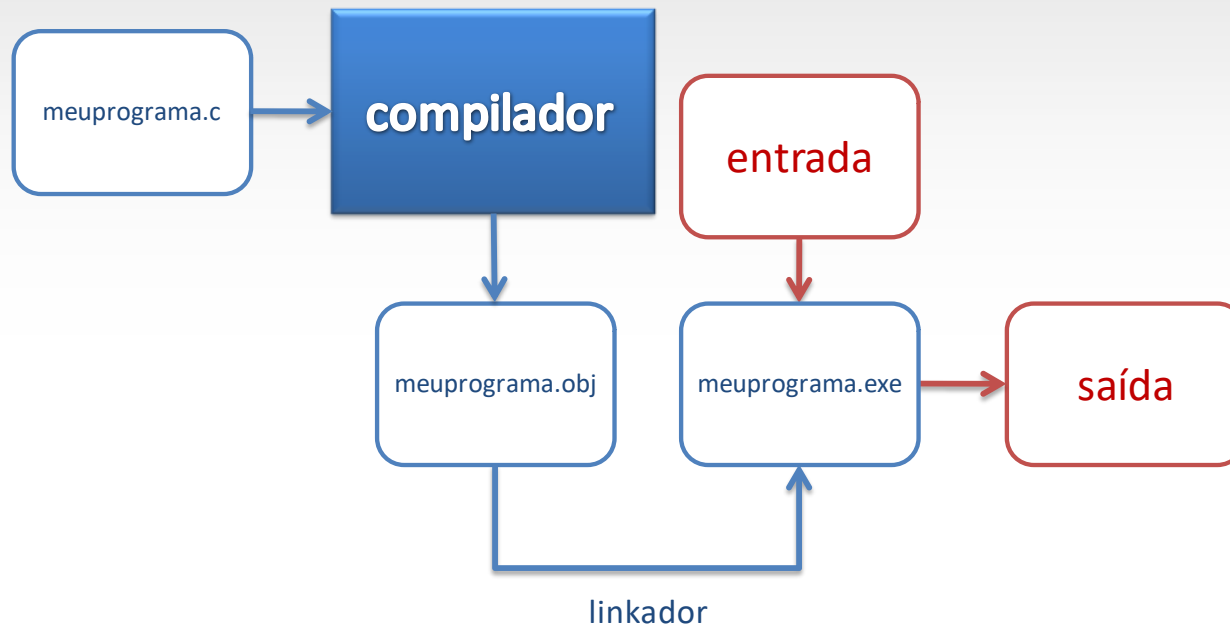


Figura 2 – Linguagens Compiladas

- ***Linguagens de Programação e Algoritmos***
 - ***Linguagens de Programação***
 - ***Exemplos (Compiladas)***
 - ***C***
 - ***C++***
 - ***Cobol***
 - ***Delphi (Object-Pascal)***
 - ***Fortran***
 - ***Pascal***
 - ***Visual Basic***

Figura 2 – Linguagens Compiladas

- ***Linguagens de Programação e Algoritmos***
 - ***Linguagens de Programação***
 - ***interpretadas***

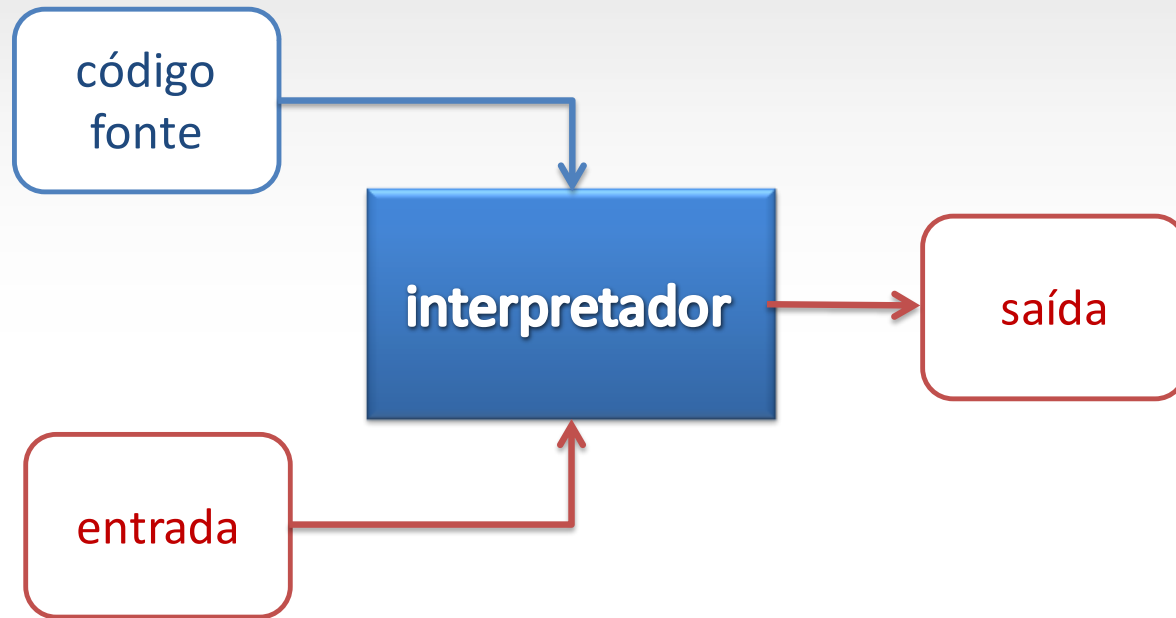


Figura 3 – Linguagens Interpretadas

- ***Linguagens de Programação e Algoritmos***
 - ***Linguagens de Programação***
 - ***Exemplos (Interpretadas)***
 - ***Basic***
 - ***Perl***
 - ***PHP***
 - ***Python***
 - ***Lisp***

- ***Linguagens de Programação e Algoritmos***
 - ***Evolução das Linguagens de Programação***











May 2022	May 2021	Change	Programming Language		Ratings	Change
1	2	▲		Python	12.74%	+0.86%
2	1	▼		C	11.59%	-1.80%
3	3			Java	10.99%	-0.74%
4	4			C++	8.83%	+1.01%
5	5			C#	6.39%	+1.98%
6	6			Visual Basic	5.86%	+1.85%
7	7			JavaScript	2.12%	-0.33%
8	8			Assembly language	1.92%	-0.51%
9	10	▲		SQL	1.87%	+0.16%
10	9	▼		PHP	1.52%	-0.34%

Figura 4 – Evolução das linguagens
Fonte: <https://www.tiobe.com/tiobe-index/>

- ***Linguagens de Programação e Algoritmos***
 - ***Métodos de Programação***

A elaboração de programas complexos requer a utilização de um método sistemático de programação que permita obter programas confiáveis, flexíveis e eficiente”

Salvetti(1998)

- ***Linguagens de Programação e Algoritmos***
 - ***Métodos de Programação***
 - *Dessa forma, sugere-se **adotar** um **método** de **programação** em **etapas**, sendo elas:*
 - *Análise do **problema** a ser **resolvido***
 - *Projeto do programa que **resolverá** o **problema***
 - » *Elaboração do **algoritmo** e **definição** das **estruturas de dados***
 - *Implementação (codificação) do programa*
 - *Testes*

- ***Linguagens de Programação e Algoritmos***

- ***Algoritmos***

- ***Solução computacional*** para um dado ***problema*** pode ser ***expressa*** por meio de ***algoritmos***
 - Assim, ***algoritmos*** são ***sequências*** de ***instruções*** e ***operações*** que facilitam a ***visualização*** da ***lógica*** de ***execução*** de uma ***solução computacional***
 - ***Os algoritmos são totalmente desprendidos de uma linguagem de programação específica***
 - O ***pseudocódigo*** é uma das ***formas*** de ***representar algoritmos***

- ***Linguagens de Programação e Algoritmos***
 - ***Fundamentos da Elaboração de Algoritmos***
 - ***Variáveis***
 - ***Constantes***
 - ***Tipos de Dados***
 - ***Expressões***

Operação	Operador
Adição	+
Subtração	-
Divisão	/
Multiplicação	*
Resto	%
Relação	Operador
Menor	<
Igual	=
Maior	>
Diferente	!=
Menor ou igual	<=
Maior ou igual	>=

- ***Linguagens de Programação e Algoritmos***
 - ***Tipos de Dados***

Tipo de dado	Descrição
inteiro	Representa o conjunto dos números inteiros na matemática.
real	Representa o conjunto dos números reais (com casas decimais) na matemática.
texto	Representa sequências de caracteres alfanuméricos que constituem um texto, uma frase, uma palavra etc.
booleano	Representa valores que somente podem ser de dois tipos: verdadeiro ou falso.
longo	Representa números grandes que não podem ser representados pelos números inteiros.

- ***Estruturas de Controle***

- ***Estruturas Condicionais***

- ***Utilizadas para tomadas de decisões ao longo dos algoritmos***
 - ***Se***
 - ***Avalie***

```
se (quantidade < 100) então  
    valorTotalAPagar ← valorTotalAPagar * 0.95;  
senão  
    valorTotalAPagar ← valorTotalAPagar * 0.8;  
fim-se
```


- ***Estruturas de Controle***

- ***Estruturas Condicionais***

- ***Utilizadas para tomadas de decisões ao longo dos algoritmos***
 - ***Se***
 - ***Avalie***

```
avale (tipoDeHabilitacao)
```

```
  caso "A": escreva("Valor da Habilitação para motos: R$ 580,00");  
    saia;
```

```
  caso "B": escreva("Valor da Habilitação para carros: R$ 650,00");  
    saia;
```

```
  caso "C": escreva("Valor da Habilitação para caminhões: R$ 1800,00");  
    saia;
```

```
  padrão: escreva("Você informou uma categoria de habilitação inválida");  
    saia;
```

```
fim-avale
```

- ***Estruturas de Controle***

- ***Estruturas de Repetição***

- ***Utilizada para repetição de determinados blocos do algoritmo***
 - ***Para...até***
 - ***Repita...até***
 - ***Enquanto...faça***

```
para i ← 1 até 1 > numero  
    fatorial ← fatorial * i;  
fim-para
```

- ***Linguagem de Programação e Algoritmos***
 - ***Modularização de Algoritmos***
 - ***Função***
 - *É um módulo que sempre **retorna** uma resposta*
 - ***Procedimento***
 - *Não retorna nenhuma resposta*

função calculaMedia(soma : inteiro, quantidade : inteiro) : real

Declare

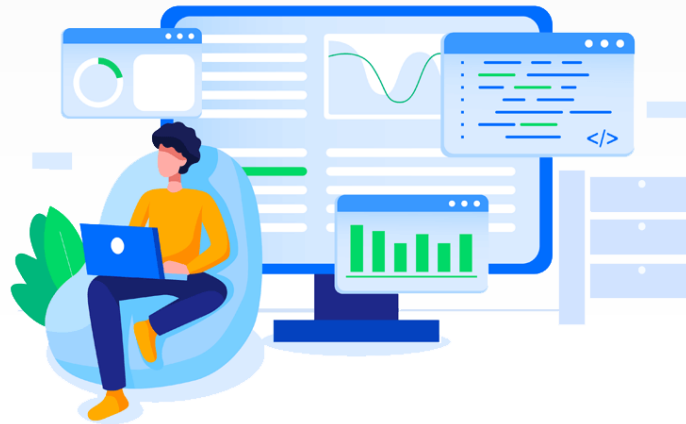
media : real; {variável com escopo de função}

Início

media \leftarrow soma / quantidade;

retorne media;

Fim



PARADIGMA DE PROGRAMAÇÃO IMPERATIVO

Imperativo

- ***Paradigma de Programação Imperativo***
 - *É um dos **primeiros paradigmas** criados e, até hoje, é bastante utilizado em diversas áreas da computação*
 - *Esse **paradigma** é estruturado com base em métodos (**procedimentos** ou **funções**)*
 - ***Conceito fundamental: o estado de um programa***
 - *O **estado** pode ser **alterado** pela **manipulação** das **variáveis** a partir dos seus **métodos***

Imperativo

- *Paradigma de Programação Imperativo*

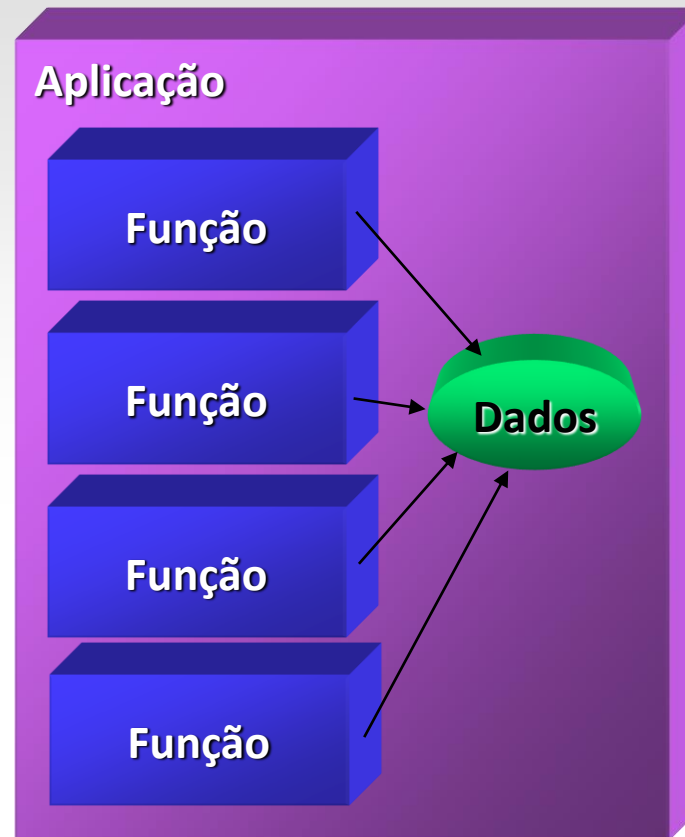


Figura 5 – Paradigma de Programação Imperativo

- *Paradigma de Programação Imperativo*

- *Vantagens*

- *Eficiência na execução dos programas*
 - *Facilidade de modelagem e modularização*

- *Desvantagens*

- *Problemas de legibilidade do código-fonte*
 - *Possibilidade de introdução de erros durante o processo de manutenção*

Imperativo

- ***Paradigma de Programação Imperativo***
 - ***Exemplo:***

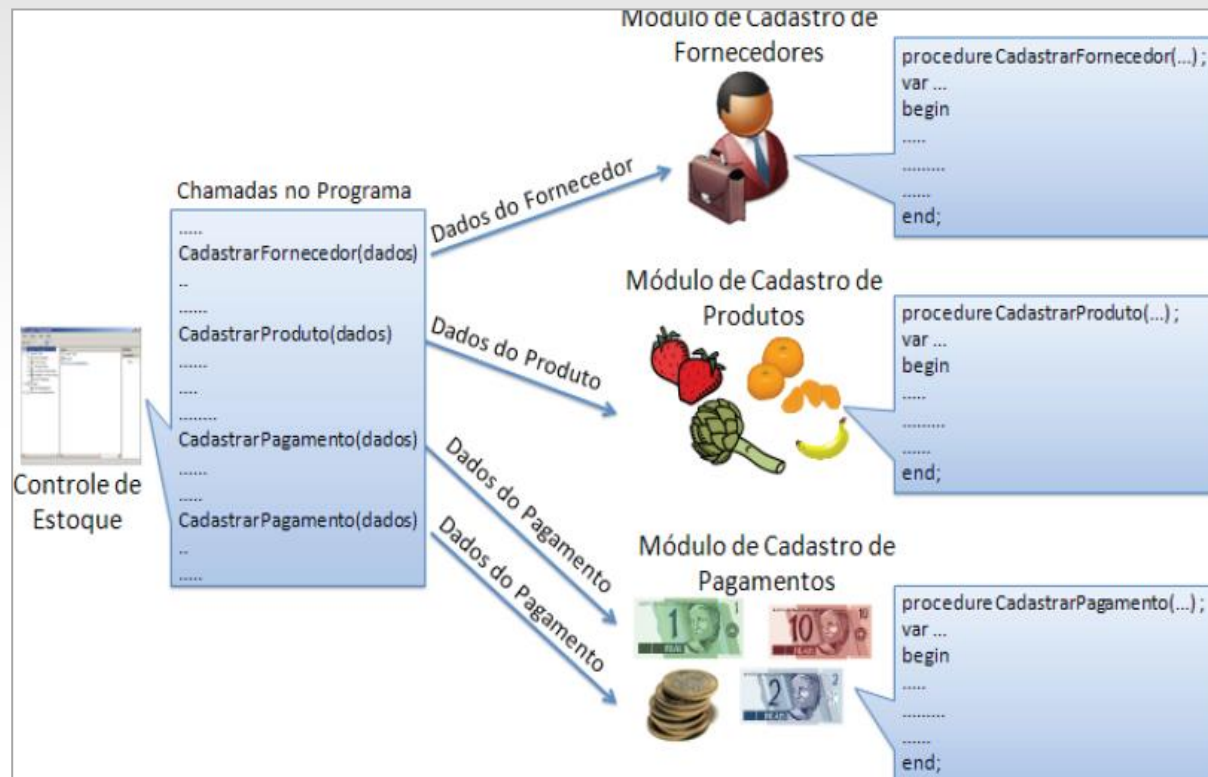


Figura 6 – Exemplo Paradigma de Programação Imperativo

Imperativo

- ***Paradigma de Programação Imperativo***
 - ***Linguagens Imperativas***
 - *Fortran, Cobol, C, Visual Basic e Pascal*
 - ***Estrutura de um Programa: Pascal***

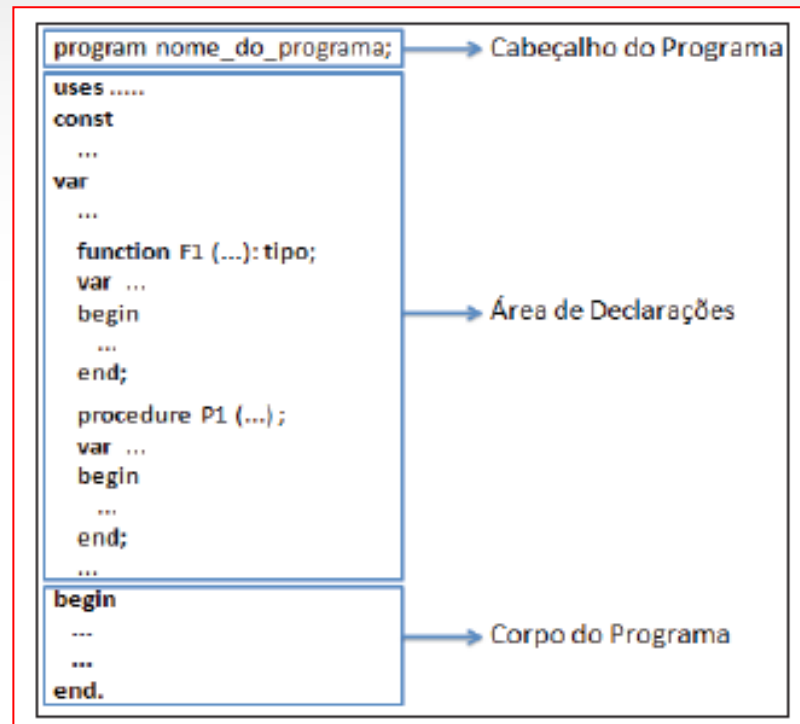
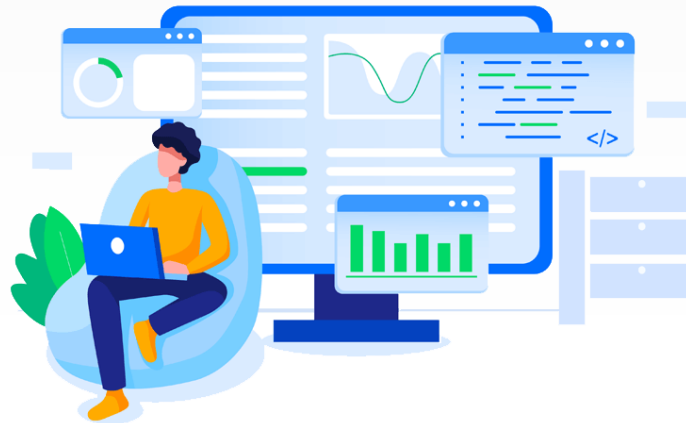


Figura 7 – Estrutura Linguagem Pascal

- **Exemplo: Linguagem Pascal**
 - **Tipos de Dados**

TIPO	REPRESENTAÇÃO NA LINGUAGEM	VALORES POSSÍVEIS
inteiro	shortint	Valores entre -128 e 127
	integer	Valores entre -32768 e 32767
	longint	Valores entre -2.147.483.648 e 2.147.483.647
	byte	Valores entre 0 e 255
	word	Valores entre 0 e 65.535
real	real	Valores entre $2,9 \times 10^{-39}$ e $1,7 \times 10^{38}$
	single	Valores entre $1,5 \times 10^{-45}$ e $3,4 \times 10^{38}$
	double	Valores entre $5,0 \times 10^{-324}$ e $1,7 \times 10^{308}$
	extended	Valores entre $3,4 \times 10^{-4932}$ e $1,1 \times 10^{4932}$
	comp	Valores entre $-9,2 \times 10^{18}$ e $9,2 \times 10^{18}$
caractere	char	Um único caractere pertencente ao conjunto de caracteres alfanuméricos (0...9, A...Z, a...z) ou especiais (exemplos: #, ?, !, @)
	string[n] *	Um conjunto de caracteres pertencentes ao conjunto de caracteres alfanuméricos ou especiais
booleano	boolean	true ou false
*Observação: o n, em string[n], especifica o número de caracteres que compõe a string.		

Tabela 1 – Data Types | Linguagem Pascal



PARADIGMA DE PROGRAMAÇÃO ORIENTADO A OBJETOS

Objetos

- **Paradigma Orientado a Objetos**

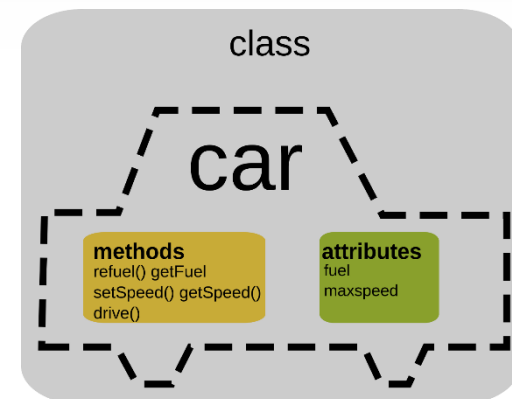
- As linguagens **orientadas** a **objetos** são baseadas em princípios que procuram **mapear** o mundo real dos objetos em programas de computador

- **Conceitos Fundamentais:**

- **Abstração, Encapsulamento, Classes, Objetos, etc.**

- **Principal Objetivo:**

- **Reutilização de código-fonte**



- *Paradigma Orientado a Objetos*

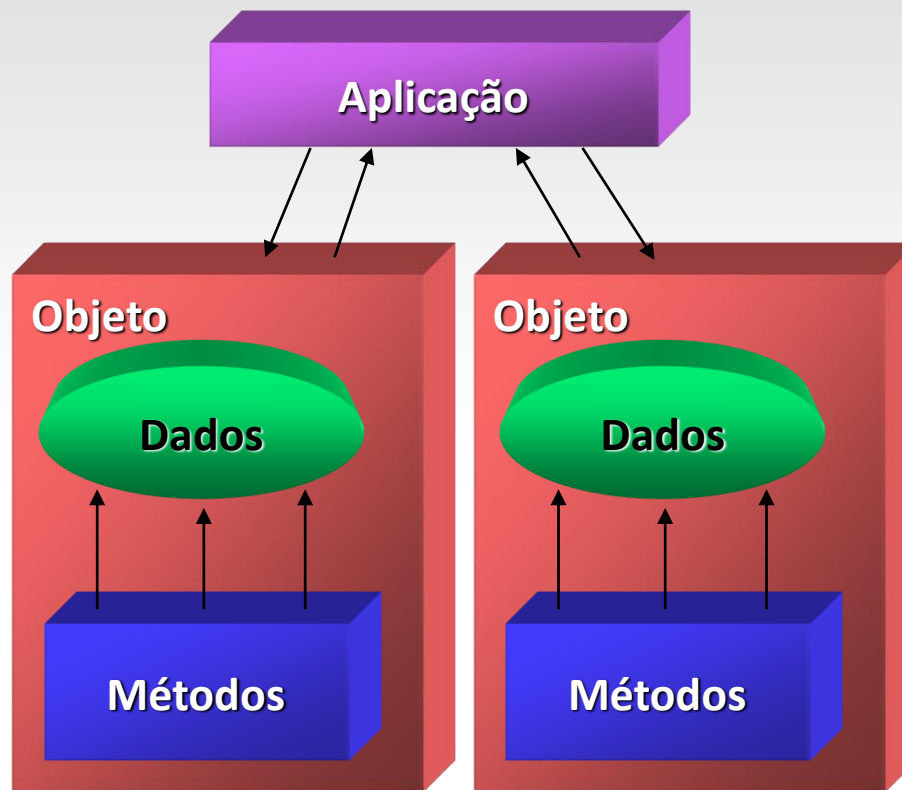
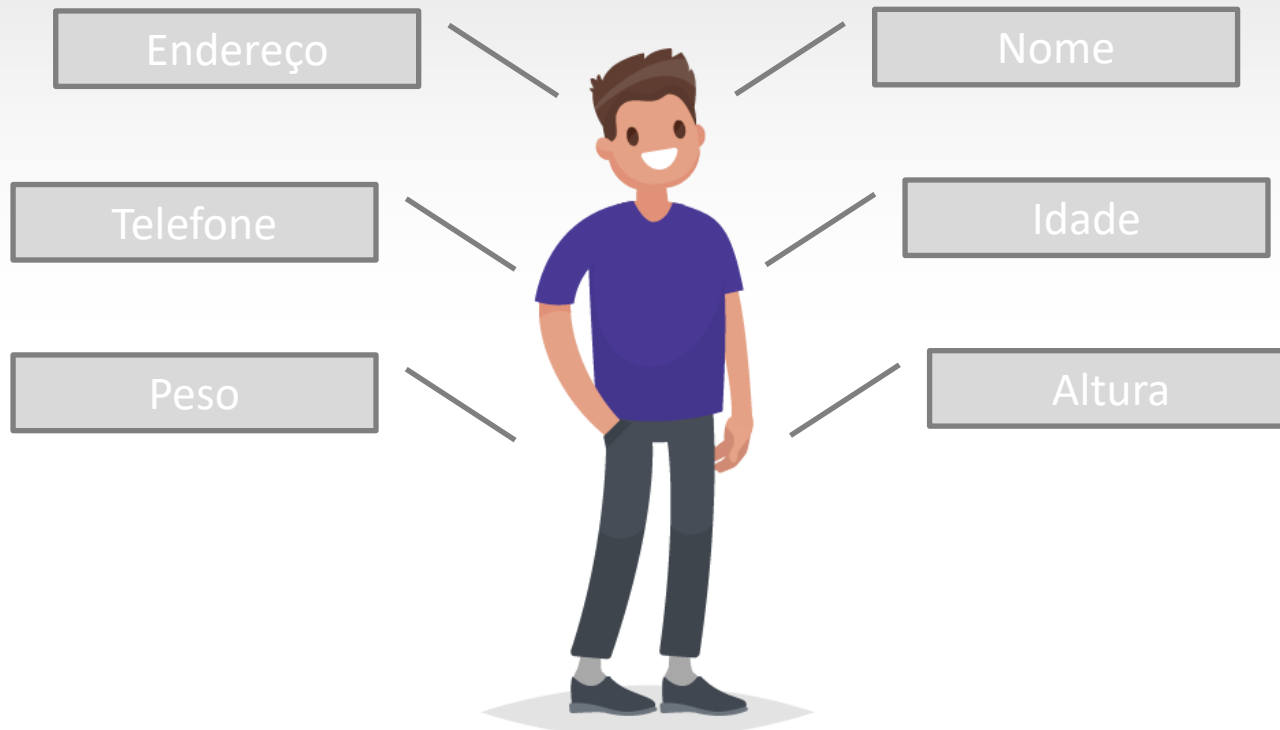


Figura 8 – Paradigma Orientado a Objetos

- **Fundamentos OO**

- **Abstração**

- **Mapeamento das informações que serão utilizadas na aplicação**



- ***Fundamentos OO***

- ***Abstração***

- ***Mapeamento das informações que serão utilizadas na aplicação***



Cliente
<ul style="list-style-type: none">- nome : texto- endereço : texto- telefone : texto- Idade : inteiro

- **Fundamentos OO**

- **Encapsulamento**

- Tem como **objetivo agrupar os conjuntos de objetos** com seus **dados** e **comportamentos** em uma **unidade** denominada **classe**



Cliente
<ul style="list-style-type: none">- nome : texto- endereço : texto- telefone : texto- Idade : inteiro
<ul style="list-style-type: none">- setNome(nome : texto)- getNome() : texto- setEndereco(endereco : texto)- getEndereco() : texto- ...

- **Fundamentos OO**

- **Classes**

- *Modelos ou especificações para construção de objetos*
 - *Representam uma descrição abstrata e genérica que permite compreender o comportamento do objeto*

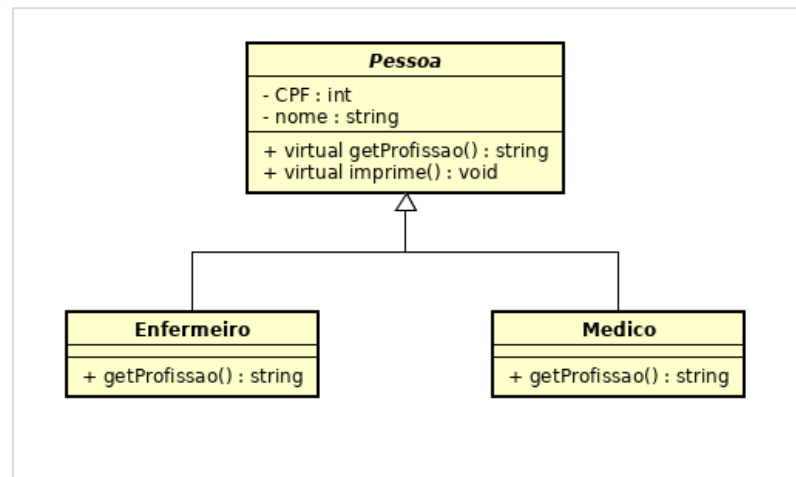
- **Objetos ou Instâncias**

- *“Criação” dos objetos a partir das especificações da classe*
 - *Contém atributos e métodos*

- **Fundamentos OO**

- **Herança**

- É uma **forma de reutilização de software** em que as **classes são criadas a partir de classes já existentes**
 - As **novas classes “herdam”** os **atributos** e **comportamentos** das **classes previamente existentes**
 - **Permitindo o aprimoramento com novos recursos**



- **Fundamentos OO**

- **Herança**

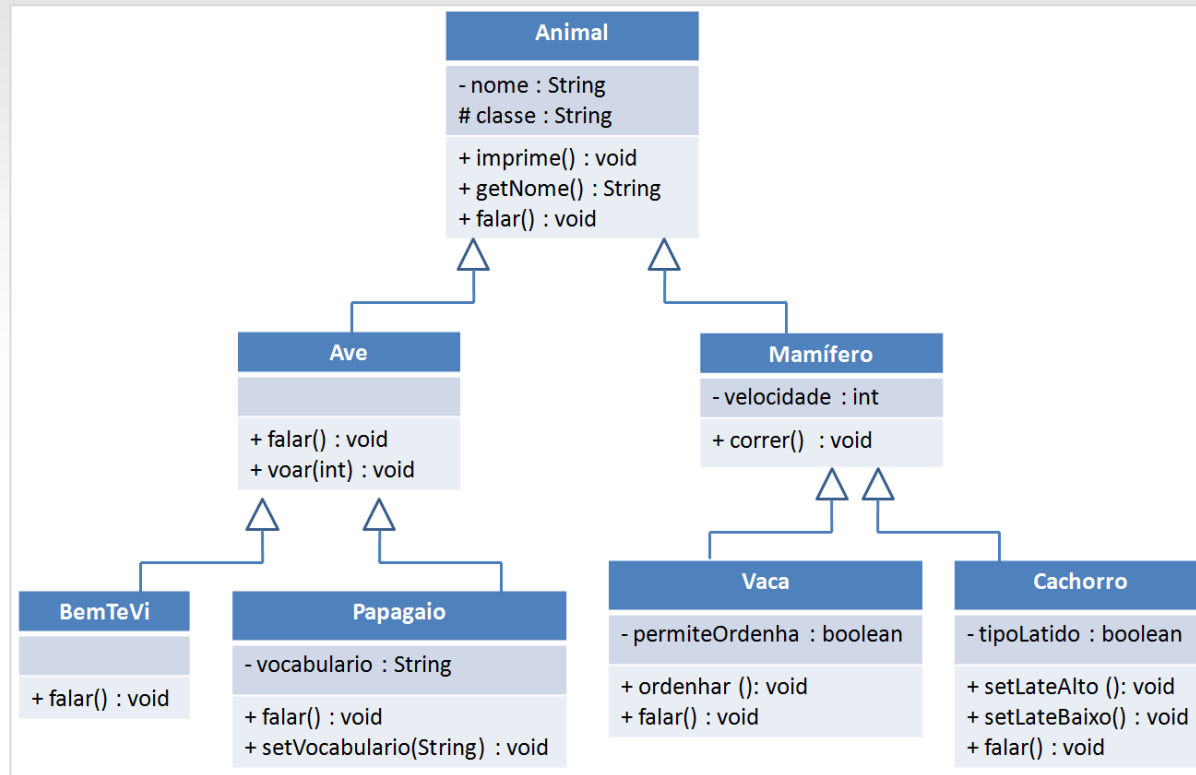
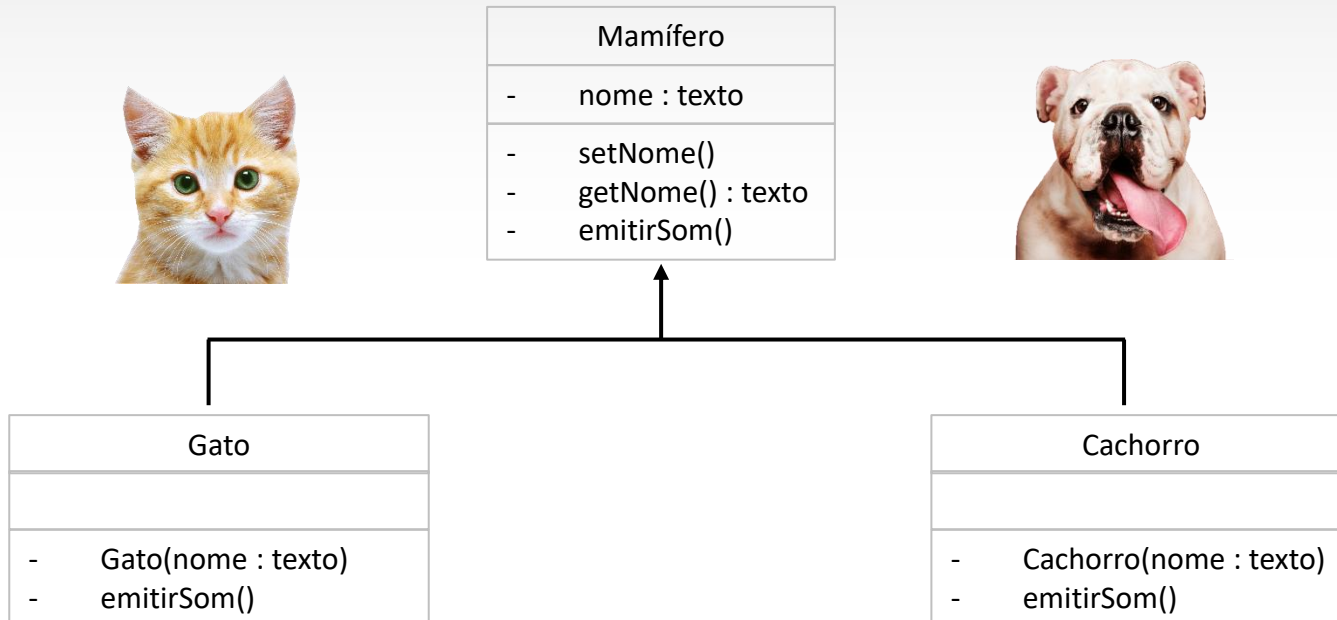


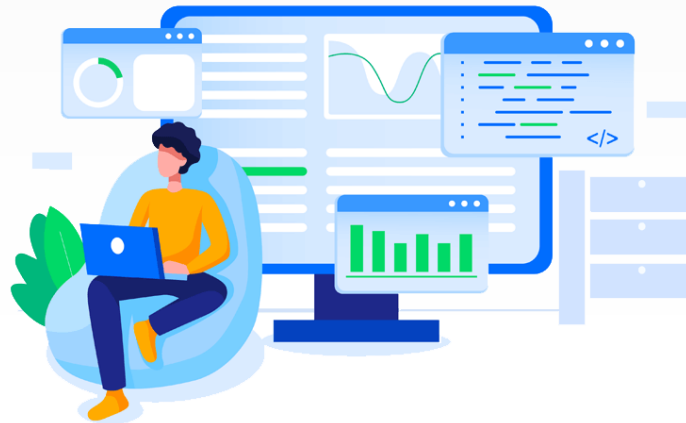
Figura 9 – Herança

- **Fundamentos OO**

- **Polimorfismo**

- Conceitualmente **significa** “*muitas formas*” e **representa** o **fato** de uma **determina característica** ser **diferente** para **cada filho**





PARADIGMA DE PROGRAMAÇÃO CONCORRENTE

Concorrente

- ***Paradigma de Programação Concorrente***
 - *Tem como **objetivo** permitir o desenvolvimento de soluções computacionais que **executam** várias **tarefas** ou **processos** ao **mesmo tempo***
 - *Uma **tarefa** consiste em uma unidade de programa que pode ser **executada** de **forma concorrente** com outras unidades*
 - *Um **processo** é um programa executável que é formado por várias tarefas*

- ***Paradigma de Programação Concorrente***
 - ***Vantagens***
 - ***Aumento de Desempenho***
 - ***Várias tarefas executadas ao mesmo tempo***
 - ***Aumento da Confiabilidade para aplicações críticas***
 - ***Em que a execução em um único processo **não** é confiável***
 - ***Implementação de sistemas distribuídos***



Concorrente

- ***Nível de Concorrência***
 - ***Instrução de Máquina***
 - ***Duas ou mais instruções de máquina executando ao mesmo tempo***
 - ***Tarefa***
 - ***Duas ou mais tarefas de um processo executando ao mesmo tempo***
 - ***Programa***
 - ***Dois ou mais processos executando ao mesmo tempo***



Concorrente

- ***Tipo de Concorrência***

- ***Lógica***

- ***Um único processador executa, de maneira intercalada, vários processos ou tarefas***

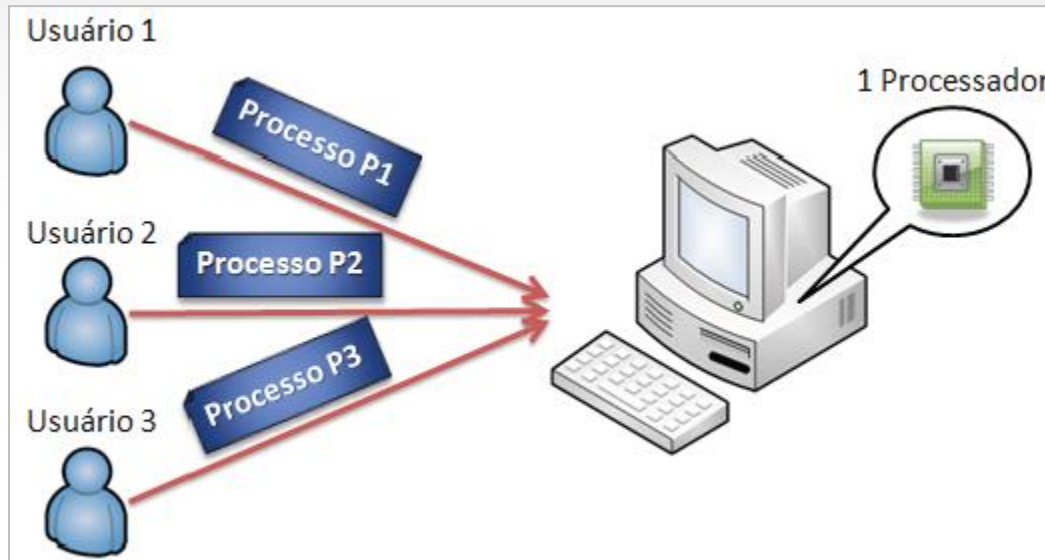


Figura 10 – Concorrência Lógica

Concorrente

- ***Tipo de Concorrência***

- ***Física***

- ***Processos ou tarefas são executados, simultaneamente, em processadores diferentes***

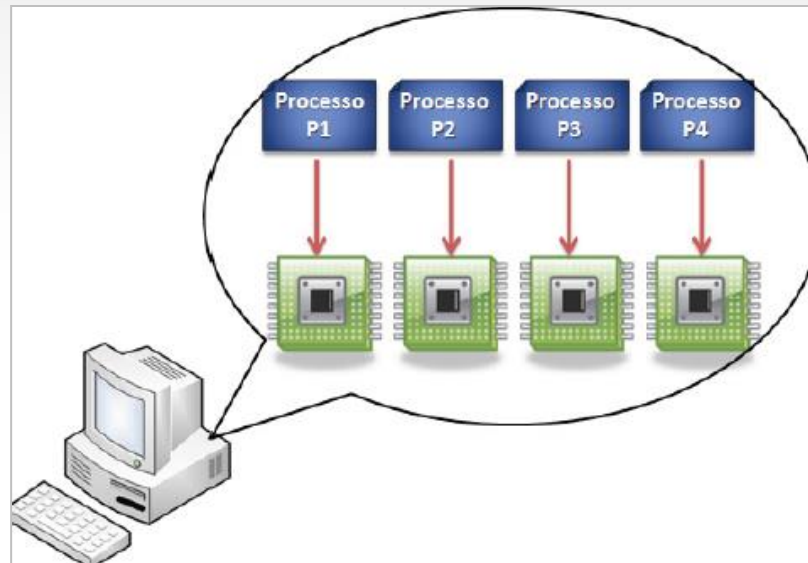


Figura 11 – Concorrência Física Multiprocessador

Concorrente

- ***Tipo de Concorrência***

- ***Física***

- ***Processos ou tarefas são executados, simultaneamente, em processadores diferentes***

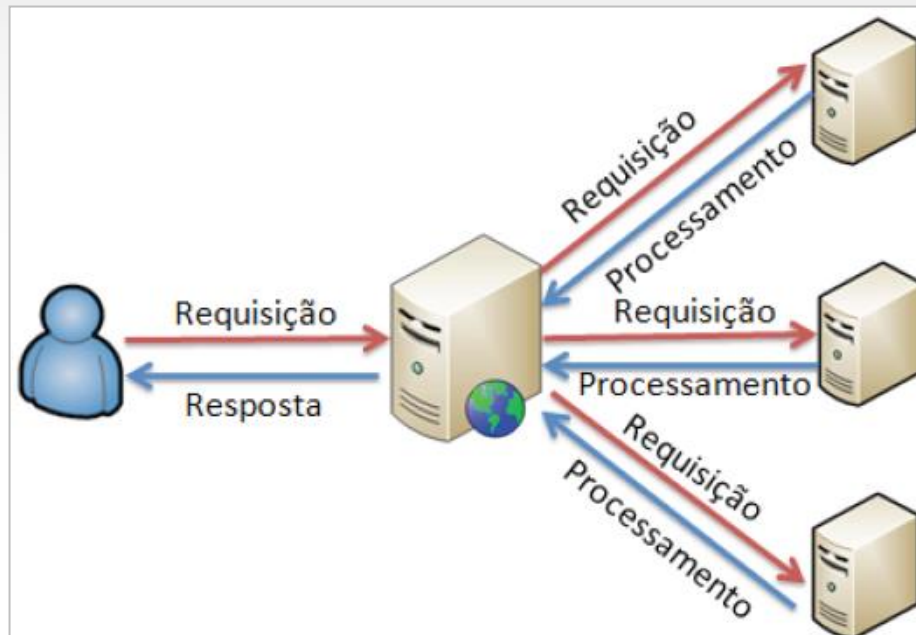
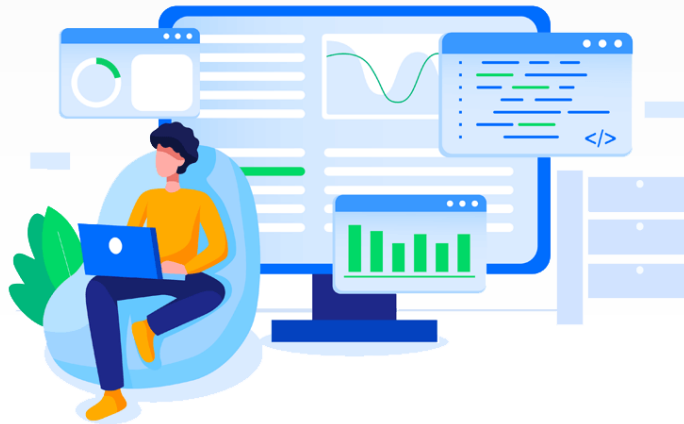


Figura 12 – Concorrência Física Multicomputador
Aula 01 | Módulo Básico



PARADIGMA DE PROGRAMAÇÃO FUNCIONAL

- ***Paradigma de Programação Funcional***
 - ***Baseado em funções matemáticas***
 - ***Enfoque principal na aplicação de funções que pode ser***
 - ***Primitivas: providas pela própria linguagem***
 - ***Compostas: formadas a partir de funções primitivas***

- ***Paradigma de Programação Funcional***
 - ***Linguagem Imperativa***
 - ***Resultado de uma expressão é armazenado em uma variável***
 - ***Linguagem Funcional***
 - ***Não contém variáveis e instruções de atribuição***
 - ***Repetições são criadas através de recursão***
 - ***Programas possuem poucas linhas de códigos***

- ***Paradigma de Programação Funcional***
 - ***Linguagens funcionais são interpretadas***
 - ***Os interpretadores operam como uma calculadora, lendo expressões, calculando os seus resultados e, em seguida, exibindo-os na tela***

- ***Paradigma de Programação Funcional***
 - ***Linguagens LISP***
 - ***Principal representante do paradigma funcional***
 - ***Muito utilizada na área de Inteligência Artificial***
 - ***Implementação específica para AutoCAD → AutoLISP***

- *Paradigma de Programação Funcional*
 - *Ciclo de Execução (LISP)*
 - *Leitura*
 - *Avaliação*
 - *Escrita*
 - O *interpretador lê* o que o usuário digita, *avalia* as expressões e *imprime* os resultados

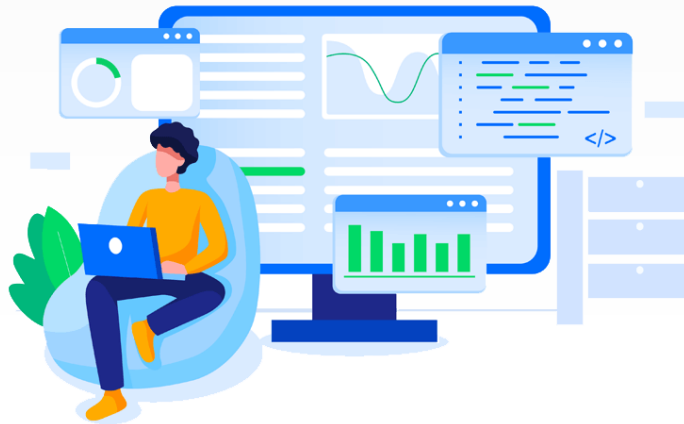
- *Paradigma de Programação Funcional*
 - *Exemplo (LISP)*

```
;;; Este programa faz a multiplicação de dois números
```

```
(setq x 5)  
(setq y 8)  
(defun mult (op1 op2)  
  (* op1 op2)  
)  
(mult x y)
```

Saída:

40

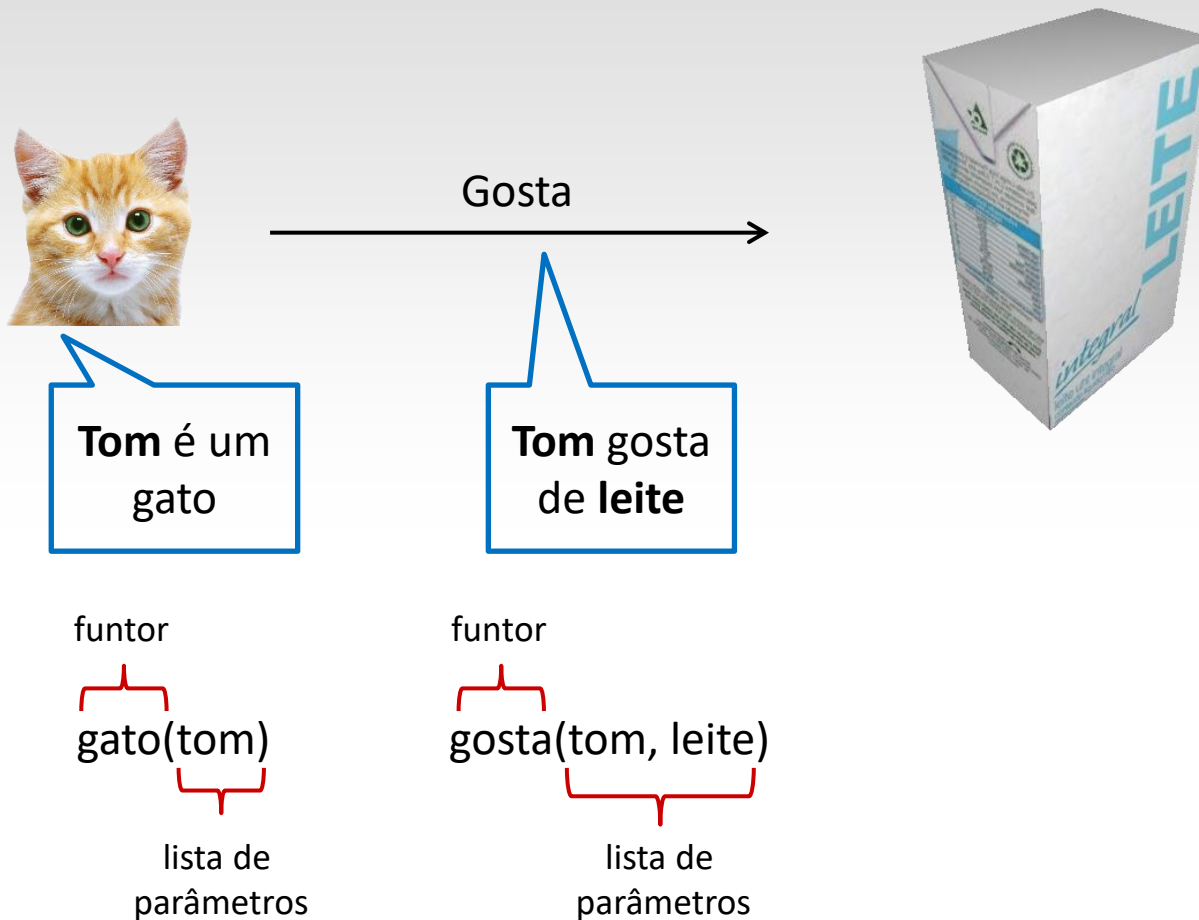


PARADIGMA DE PROGRAMAÇÃO LÓGICA

- **Paradigma de Programação Lógica**
 - Também conhecido como **paradigma declarativo** ou **simbólico**
 - Possui como **base** a **lógica formal**, a qual trata a relação entre **conceitos** e **fornece mecanismos para compor provas de proposições**
 - A **programação** é definida a partir de **proposições simples** chamadas de **proposições atômicas** e consistem em **termos compostos**

- ***Paradigma de Programação Lógica***
 - ***Um termo composto possui duas partes:***
 - ***Funtor:*** *corresponde ao símbolo da função que nomeia uma relação*
 - ***Lista Ordenada:*** *define a lista de parâmetros passados ao funtor*

- *Paradigma de Programação Lógica*



- **Paradigma de Programação Lógica**
 - Como é **baseado na lógica** podemos utilizar **conectivos**

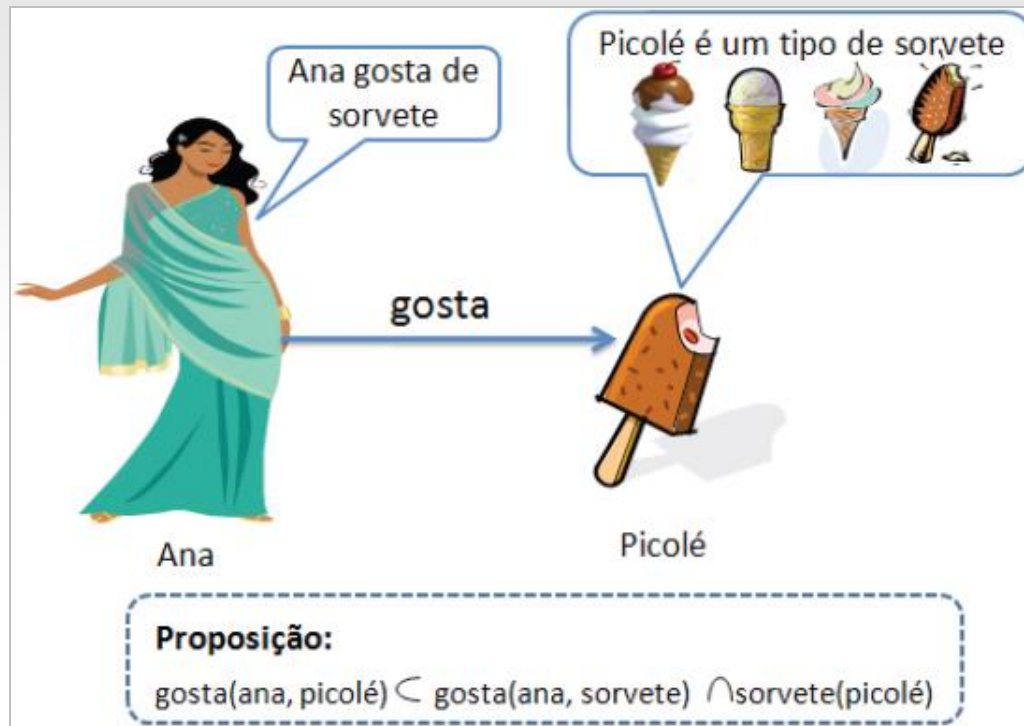


Figura 13 – Proposição utilizando operadores de implicação e conjunção

- **Paradigma de Programação Lógica**
 - Como é **baseado na lógica** podemos utilizar **conectivos**

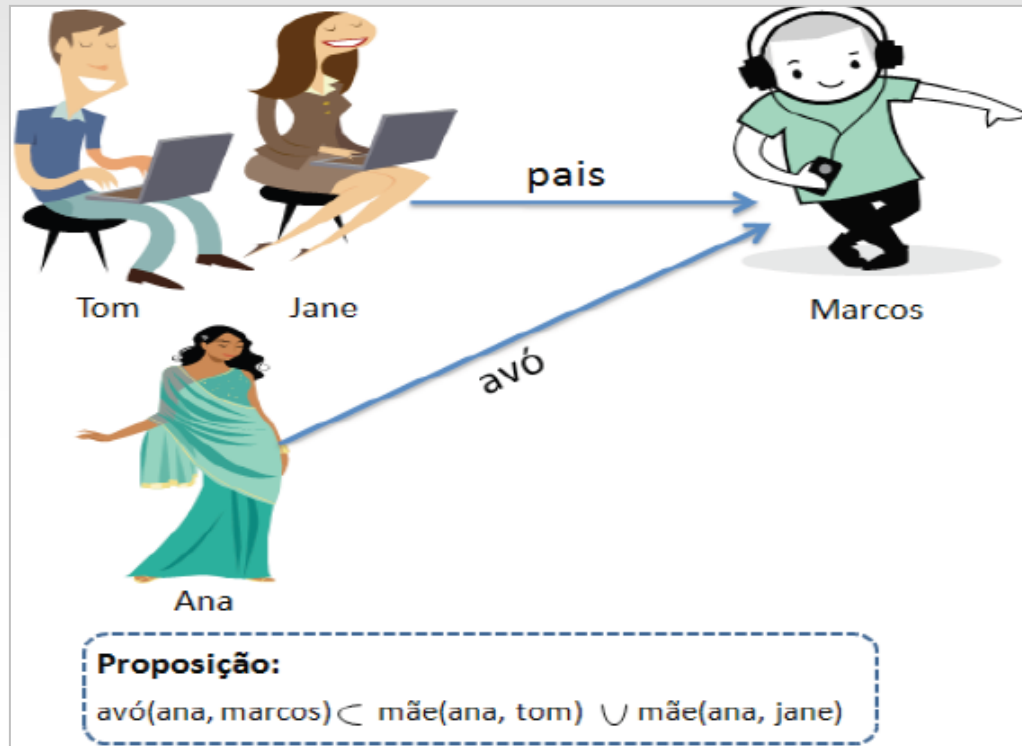


Figura 14 – Proposição utilizando operadores de implicação e disjunção

- ***Paradigma de Programação Lógica***
 - ***Prolog***
 - ***Principal linguagem de programação lógica***
 - ***Exemplo:***

Fatorial de um número

```
%comentários são precedidos por %  
%este programa permite calcular o fatorial de um número  
fatorial(0, 1): -! .  
fatorial(0, 1): -! .  
Fatorial(X, N):- X2 is X-1, fatorial(X2, N1), N is X * N1
```

Consulta:

?- fatorial(5, Resultado).

Resultado = 120

Referências

Melo, A. C. V.; Silva, F. S. C. *Princípios de Linguagens de Programação*. São Paulo: Edgard Blücher Ltda., 2003.

Sebesta, Robert W. *Conceitos de Linguagens de Programação*. Tradução de José Carlos Barbosa dos Santos. 5. ed. Porte Alegre: Bookman, 2003.

Tucker, A.; Noonam, R. *Linguagens de Programação: Princípios e Paradigmas*. 2. ed. São Paulo: McGrawHill, 2009.