Linguagens de Programação: Introdução

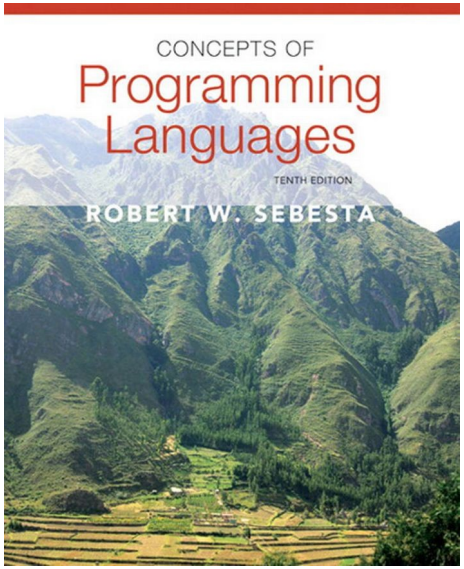
Prof. Arnaldo Candido Junior

UNESP – IBILCE

São José do Rio Preto, SP

Ementa

- Tópicos...
- Bibliografia…
- Calendário...
- Google Classroom: tmk2a23
- Notas
  - NF = 0.16AP + 0.42P1 + 0.42P2
  - NF’ = 0.5NF + 0.5E



Objetivo

- Estudar os quatro grandes paradigmas para desenvolvimento de linguagens de programação
  - Imperativo
  - Lógico (**foco**)
  - Funcional (**foco**)
  - Orientado à objetos

Parte teórica da disciplina

- Disciplina Linguagens de Programação: como **projetar** uma linguagem nova?
  - Disciplina principalmente teórica
  - Mas vamos incluir algumas coisas práticas (a seguir)
- Disciplina Compiladores: como **implementar** uma linguagem?

Parte Prática

- Paradigma **Imperativo**:
  - Sintaxe avançada (ex.: ambiguidade do if)
  - Conceitos avançados (ex.: efeitos colaterais, mudança de estado, operadores em curto circuito)
- Paradigma **Lógico**: introdução a Prolog

Parte Prática (2)

- Paradigma **Funcional**: introdução à Lisp
- Paradigma **Orientado a Objetos**: boas práticas
- Conceitos especiais:
  - Estilos de execução
  - Estilos de gerenciamento de memória
  - Tendências em linguagens de programação

Paradigma

- “Algo que serve de exemplo ou modelo; padrão”
  Dicionário Michaelis

- É como padrão, modelo ou exemplo a ser seguido a respeito de algo
- Podemos pensar que é uma forma de ver um problema ou abordar um problema

Paradigma (2)

- **Paradigma de linguagem de programação**:
  - Estratégia para abordar o problema de se criar, projetar e implementar uma linguagem de programação
  - É um ponto de vista sobre como a linguagem deve ser implementada
  - Cada paradigma de linguagem de programação tem um conceito central em foco

Paradigma (3)

- Uma Linguagem de Programação é uma forma eficiente de comunicação entre humanos (técnicos) e a máquina (computador)
  - Nem baixo nível (linguagem de máquina)
  - Nem altíssimo nível (língua humana)

Paradigma (4)

- Razões para estudar paradigmas
  - Capacidade aumentada para expressar ideias
  - Base melhor para decidir qual linguagem utilizar
  - Entender a importância da implementação: afeta várias decisões no projeto de uma linguagem
  - Continua..

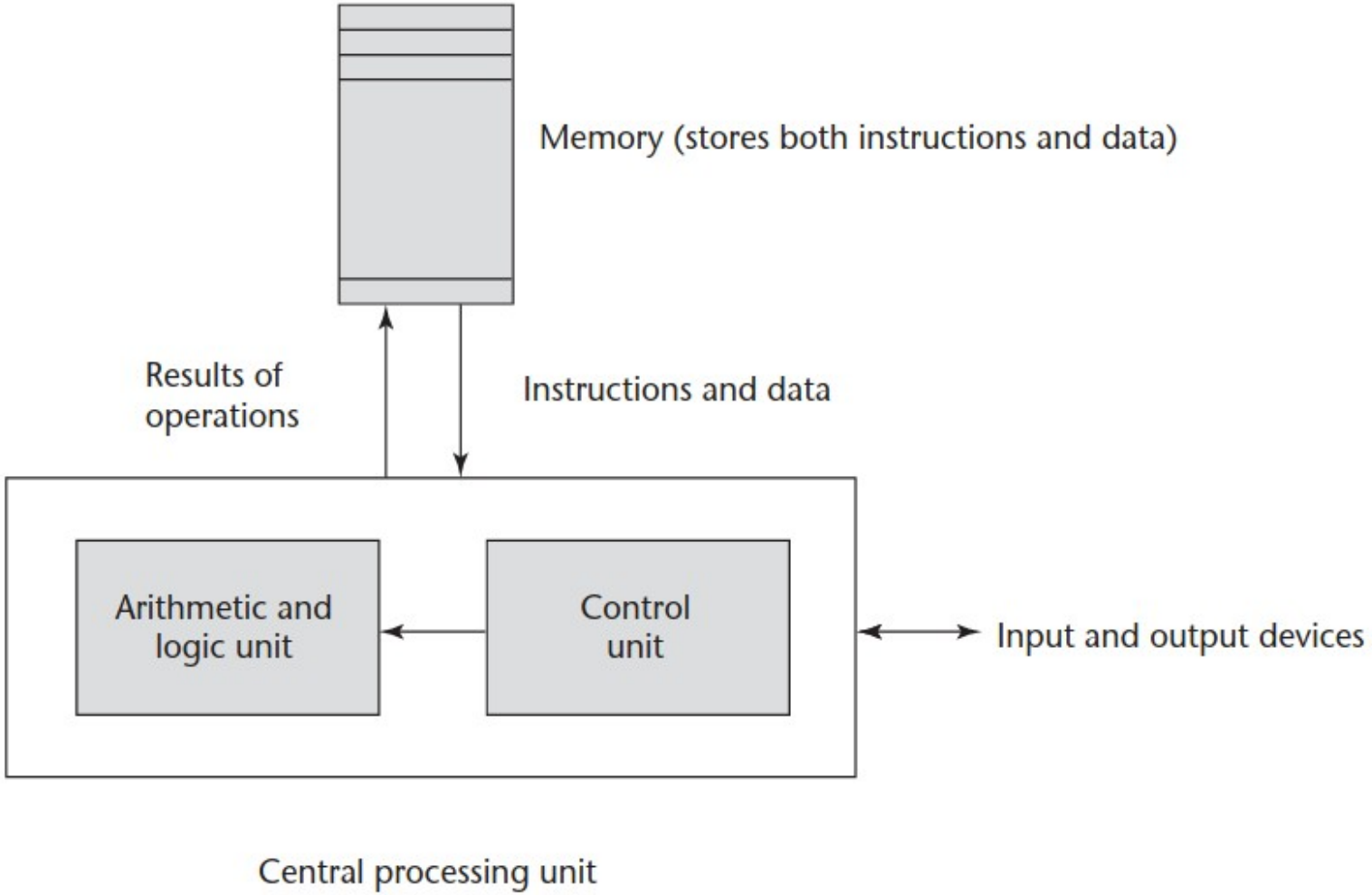
Paradigma (5)

- … continuação:
  - Usar melhor as linguagens que você já conhece
  - Entendimento dos avanços na computação: as vezes nem sempre a “melhor” linguagem se torna a mais popular

Paradigma (6)

- Interação com a máquina em cada vertente deve ser feita por meio de:
  - Imperativo: comandos
  - Lógico (**foco**): declarações de fatos (informações consideradas como verdadeiras)
  - Funcional (**foco**): funções
  - Orientado à objetos: objetos que descrevem instâncias do mundo real por meio atributos e métodos

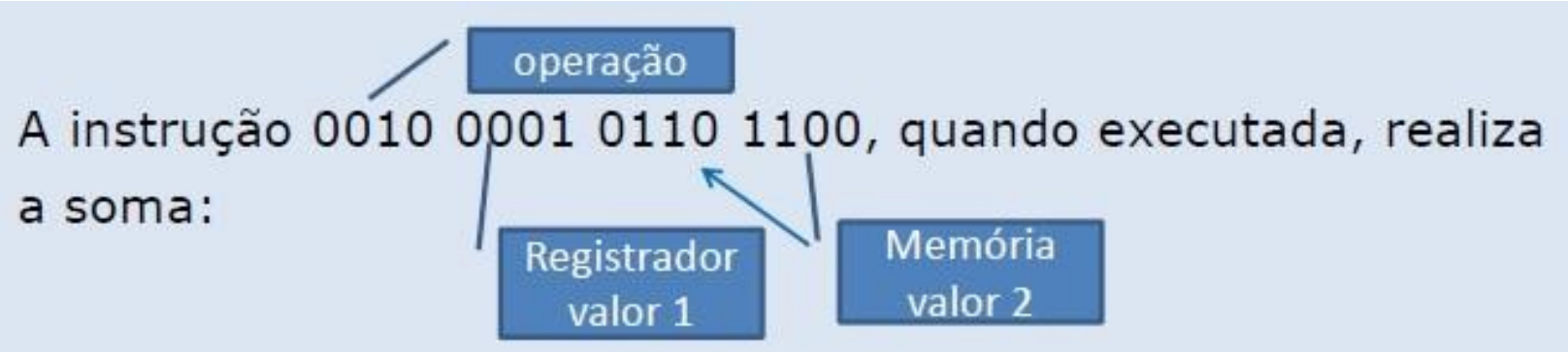
Código de máquina: Arquitetura de von Newmann



Código de máquina: Arq. de von Newmann (2)

- Máquina de **Turing**: arquitetura teórica para estudar quais tipos de problemas são computáveis
  - Resolvíveis por computador
- Arquitetura de **von Newmann**: base de todos os computadores modernos
  - Define por exemplo, que código e dados ocupam mesma memória
  - Grande impacto no projeto de todas as linguagens, independente de paradigma

Código de máquina: exemplo

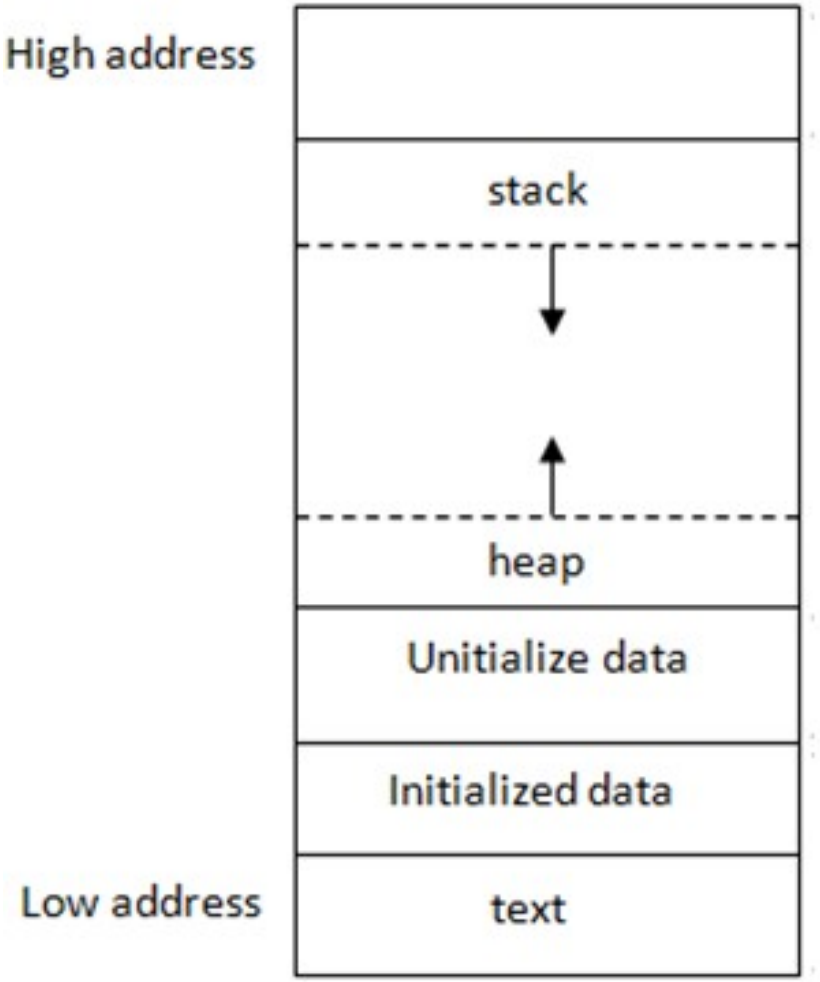


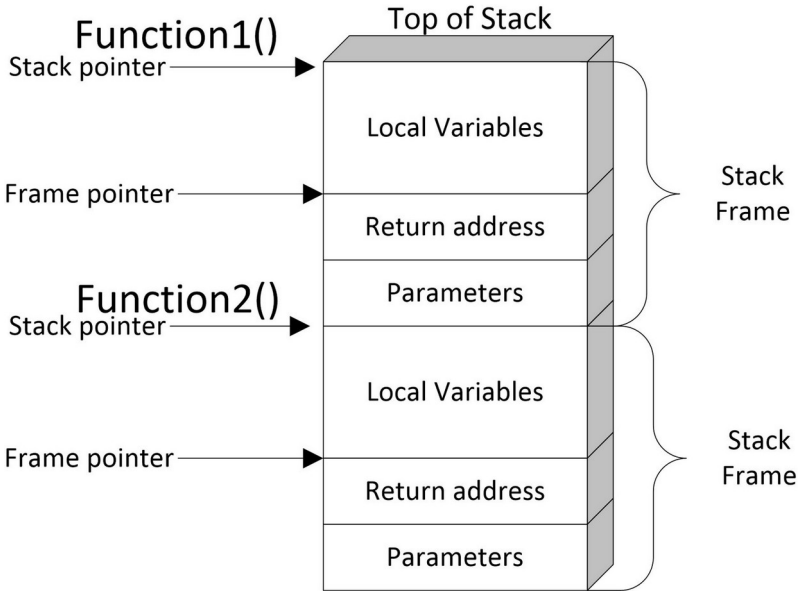
- Primeiros programas, linguagens de montagem e compiladores:
  - Escritos em código de máquina
- Exemplo prático: dump hexadecimal de código executável

Linguagem de montagem: visão geral

- A linguagem mais próxima da máquina
- Cada comando em linguagem de montagem corresponde a uma operação do processador
- Usa **mneumônicos**: nomes fáceis para memorizar as operações
  - Mais a seguir

Linguagem de montagem: heap vs stack





Linguagem de montagem: stack

- Pilha que armazena variáveis locais e globais cujo o tamanho não varia e outros dados (ponteiros de retorno de função)
- Segue a estratégia LiFo (Last In, First Out)
- Convenção, começa em posições altas de memória e avança para posições baixas
- Discutiremos brevemente para entender algumas decisões do paradigma imperativo

Linguagem de montagem: heap

- Guarda variáveis dinâmicas como arrays dinâmicas, listas encadeadas, entre outros
  - Basicamente tudo que usa malloc em C
- Cresce das posições de memória mais baixas até as mais altas
- Não é LiFo: é responsabilidade do sistema operacional gerenciar posições vagas
  - É mais custoso que a stack

Linguagem de montagem: exemplos

- Comando para gerar os exemplos a seguir
  - gcc -S -m32 -O0 -fno-asynchronous-unwind-tables arquivo.c
- Obs1: note que os exemplos foram simplificados para fins didáticos
- Obs2: note que inteiros ocupam 4 bytes de memória na arquitetura de exemplo (i386)

Exemplo 1: variáveis (C)

```c
void main () {
  int ano = 2023;
  int nascimento = 2001;
  int idade = 18;
}
```

Exemplo 1: variáveis (ASM)

```
main:
   pushl    %ebp
   movl     %esp, %ebp
   subl     $16, %esp           # até aqui: preâmbulo
   movl     $2023, -12(%ebp)    # ano
   movl     $2001, -8(%ebp)     # nascimento
   movl     $18, -4(%ebp)       # idade
   leave
   ret
```

Exemplo 2: if (C)

```c
void main () {
  int ano = 2023;
  int nascimento = 2001;
  int idade;
  if (ano < nascimento) {
    idade = 0;
  } else {
    idade = ano - nascimento;
  }
}
```

Exemplo 2: if (ASM)

```
main:
  pushl   %ebp
  movl    %esp, %ebp
  subl    $16, %esp
  movl    $2023, -12(%ebp)
  movl    $2001, -8(%ebp)
  movl    -12(%ebp), %eax
  cmpl    -8(%ebp), %eax
  jge     .L2
  movl    $0, -4(%ebp)
  jmp     .L4
.L2:
  movl   -12(%ebp), %eax
  subl   -8(%ebp), %eax
  movl   %eax, -4(%ebp)
.L4:
  leave
  ret
```

Exemplo 3: while (C)

```c
void main () {
  int ano = 2001;
  int idade = 0;
  while (ano < 2023) {
    total++;
    ano++;
  }
}
```

Exemplo 3: while (ASM)

```
main:
  pushl   %ebp
  movl    %esp, %ebp
  subl    $16, %esp
  movl    $2001, -8(%ebp)
  movl    $0, -4(%ebp)
  jmp     .L2
.L3:
  addl    $1, -4(%ebp)
  addl    $1, -8(%ebp)
.L2:
  cmpl    $2022, -8(%ebp)
  jle     .L3
  leave
  ret
```

Exemplo 4: função (C)

```c
int f(int ano, int nascimento) {
  return ano - nascimento;
}

void main () {
  int ano = 2023;
  int nascimento = 2001;
  int idade = f(ano, nascimento);
}
```

Exemplo 4: função (ASM)

```
f:
    pushl   %ebp
    movl    %esp, %ebp
    movl    8(%ebp), %eax
    subl    12(%ebp), %eax
    popl    %ebp
    ret
```

```
main:
    pushl   %ebp
    movl    %esp, %ebp
    subl    $16, %esp
    movl    $2023, -12(%ebp)
    movl    $2001, -8(%ebp)
    pushl   -8(%ebp)
    pushl   -12(%ebp)
    call    f
    addl    $8, %esp
    movl    %eax, -4(%ebp)
    leave
    ret
```

Contador de Programa

- O processador contém um registrador especial conhecido como contador de programa (Instruction Pointer)
  - Normalmente é incrementado
  - Exceto para os comandos como jmp, jge, call, ret, entre outros
- Mais sobre assembly na disciplina Laboratório de Linguagem de Montagem

Linguagem Imperativa

- Dispensa apresentações
- Vista em disciplinas anteriores
- Exemplos para linguagem de montagem foram contrastados com linguagem imperativa (C)

Linguagem Lógica: fatos, regras e perguntas

```
% fatos
planeta(terra).
planeta(marte).
Estrela(sol).

% perguntas
?- planeta(terra).
   yes
?- planeta(plutão).
   no
```

```
% fatos
pai(marcelo, alex).
pai(alex, antonio).

% regras
avo(X, Y) :-
  pai(X, Z),
  pai(Z, Y).

% perguntas
?- pai(marcelo, alex).
   yes
?- pai(alex, marcelo).
   No
```

Linguagem Lógica: fatos, regras e perguntas (2)

- Leitura informal da regra de exemplo:
  X é avô de **Y** quando
  X é pai de **Z** e
  Z é pai de **Y**
- Leitura formal:
  quaisquer que sejam **X** e **Y**,
  X é avô de **Y** se e somente existe Z tal que
  X é pai de **Z** e
  Z é pai de **Y**

Linguagem Lógica: fatos, regras e perguntas (3)

- Fato: aquilo que se assume como verdade
- Regra: sequência de passos para inferir novos fatos
  - Usa-se um tipo de **inferência lógica** conhecida como **dedução**
  - Permite a geração de novos conhecimentos a partir de conhecimentos prévios
- Pergunta: aquilo que se desejar saber se é verdade

Linguagem Lógica: inquirindo

```
% pergunta simples
?- pai(marcelo, alex).
   yes
?- pai(alex, marcelo).
   no

% pergunta com variável
?- pai(X, alex).
   X = marcelo
?- pai(marcelo, X).
   X = alex
```

Linguagem Lógica: inquirindo (2)

```
?- pai(X, Y).
   X = marcelo
   Y = alex ? ;

   X = alex
   Y = antonio

   yes

?- avo(marcelo, antonio).
   yes
?- avo(alex, marcelo).
   no
```

```
?- avo(X, Y).
   X = marcelo
   Y = antonio
```

Linguagem Funcional: 3 exemplos de hello world

```
(defun hello1 ()
  (format t "Exemplo 1 - olá mundo~%"))

(defun hello2 ()
  (let ((nome "arnaldo"))
    (format t "Exemplo 2 - olá ~a!~%" nome)))

(defun hello3 ()
  (format t "Exemplo 3 - informe o seu nome: ")
  (let ((nome (read-line)))
    (format t "Olá ~a!~%" nome)))

(defun main ()
  (hello1)
  (hello2)
  (hello3))
```

Linguagem Lógica: funções

```
(defun media (x y)
  (/ (+ x y) 2.0))

(defun main ()
  (format t "Média entre 10 e 20: ~a~%"
    (media 10 20)))
```

Linguagem Funcional: estrutura condicional

```
(defun sinal (x)
  (if (< x 0)
    "negativo"
    "positivo"))

(defun main ()
  (format t "Sinal 5: ~a~%"
    (sinal 5))
  (format t "Sinal -9: ~a~%"
    (sinal -9)))
```

Linguagem Funcional: laços

- Laços não são um recurso oficial de linguagens puramente funcionais
  - Mas podem ser emulados
- Usa-se recursão em vez de laços
  - Tail recursion: quando a expressão retornada é a última calculada, é possível limpar a stack
  - Evita o famoso erro de stack overflow

Linguagem Funcional: laços (2)

```
(defun main ()
  (loop for i from 1 to 10 do
    (print i)))
```

Linguagem Funcional: variáveis

- **Constante**: nunca muda o valor assumido
  - Conhecida em tempo de compilação
  - Exemplo: pi = 3.14
- **Variável imutável**: uma vez que assume um valor, não é mais possível alterá-lo
  - Conhecida apenas em tempo de execução
  - Continua...

Linguagem Funcional: variáveis (2)

- **Variável imutável**: continuação
  - Conhecida apenas em tempo de execução. Mais fiel ao conceito matemático de variável
  - Exemplo - uma equação de primeiro grau: x é a solução da equação, então seu valor não muda
  - Correspondente em C:
    const char c = getchr();

Linguagem Funcional: variáveis (3)

- **Variável mutável**: é tipo de variável mais comum nas linguagens imperativas clássicas
  - A grande maioria das variáveis na linguagem C
  - O valor da variável pode mudar a qualquer tempo

Linguagem Funcional: funções

- **Função pura**: não muda o estado global do programa
  - Nem evoca funções que mudam
  - São mais fiéis a definição de função dentro da matemática
- **Função impura**: muda o estado global do programa ou invoca funções que o façam

Linguagem Funcional: imutabilidade

- Estado mutável
  - Mais eficiente de modo geral
- Estado imutável
  - Mais seguro contra erros de programação
  - Mais fácil de paralelizar sem entrar em erros como deadlocks e condições de disputa

Linguagem Funcional: Lisp

- A linguagem Lisp incentiva o programador a usar funções puras e variáveis imutáveis
  - Mas permite o uso de funções impuras e variáveis mutáveis para quando seu uso fizer sentido
- É **multiparadigma**: funcional e imperativa

Linguagem Funcional: shadowing

- Importante: existe um conceito chamado **shadowing** (sombreamento)
  - Consiste na existência de duas variáveis com mesmo nome
  - Mas que correspondem a posições diferentes de memória
  - Não deve ser confundido com mutabilidade

Linguagem Orientada a Objetos

- Evolução das linguagens imperativas, baseada em partes na teoria de Frames da Inteligência Artificial
- Muito intuitivo para modelar problemas complexos do mundo real
- Por hora, vamos assumir intuitivamente que um objeto é uma struct da linguagem C
  - Mas que, além de valores (atributos), também pode conter funções (métodos)

Linguagem Orientada a Objetos (2)

- Contém alguns conceitos avançados para o momento:
  - Classes, instâncias, interfaces
  - Encapsulamento, herança, polimorfismo
  - Tipos abstratos de dados, tipos genéricos
  - Entre outros

Linguagem Orientada a Objetos (3)

- Esses conceitos serão oportunamente detalhados
  - Em partes na disciplina de Programação Orientada à Objetos
  - Em partes nesta própria disciplina
- Obs: linguagens OO são considerada no livro texto com um subtipo de imperativas.
  - Por simplificidade, consideraremos como um paradgima a parte.

Exemplo OO

```
class Animal {
    public void fala() {
        System.out.println("este animal ainda não sabe falar");
    }
}

class Cao extends Animal {
    @Override
    public void fala() {
        System.out.println("este cachorro late");
    }
}

// continua...
```

Exemplo OO (2)

```
class Gato extends Animal {
    @Override
    public void fala() {
        System.out.println("este gato mia");
    }
}

public class Exemplo {
    public static void main(String[] args) {
        Animal meuAnimal = new Animal();
        Animal pluto = new Cao();
        Animal tom = new Gato();
        meuAnimal.fala();
        pluto.fala();
        tom.fala();
    }
}
```

Exemplo OO: Java

- O exemplo apresentado está na linguagem Java:
  - Originalmente concebida para ser puramente orientada a objetos
  - Foco em design simples
  - Com o passar do tempo, incorporou alguns recursos de linguagens funcionais

Exemplo OO: polimorfismo

- Exemplo intuitivo preliminar de polimorfismo
- Uma **struct** pode dar origem a diferentes variáveis com dados diferentes
- Uma **classe** pode dar origem a diferentes objetos com **dados** e **comportamentos** diferentes
  - Classes também tem comportamentos (métodos)
  - Os objetos da mesma classe podem comportar de maneira diferente (diferentes **subclasses**)

Outros paradigmas

- Além dos 4 grandes paradigmas citados no livro texto, vale mencionar outros (sub-)paradgimas
  - Paradigma Concorrente
  - Paradigma Orientado a Eventos (caso especial de OO)
  - Paradigma Orientado a Aspectos (caso especial de OO)

Outros paradigmas (2)

- Obs: linguagens de marcação não são contempladas
  - Não projetadas para serem Turing-Completas (laços, estruturas de decisão, subprogramas)
  - Por isso não são consideradas como linguagem de programação
  - Exemplos: HTML, CSS, JSON, XML, entre outras

Nichos

- Existem nichos diferentes que as linguagens de programação
- Algumas linguagens são mais especializadas em determinados nichos
- Outras buscam atender a mais de um nicho
- Principais nichos a seguir...

Nichos (2)

- Aplicações científicas
- Sistemas comerciais
- Inteligência Artificial
- Programação de sistemas (aqui com sentido de sistema operacional e softwares de apoio)
- Desenvolvimento Web

Nichos (3)

- A classificação do livro texto é um pouco antiga. Vale apena também mencionar os seguintes nichos:
  - Desenvolvimento móvel
  - Sistemas embarcados

Abstração

- Programar exige pensando **abstrato**
- Abstração: “considerar isoladamente alguns aspectos específicos de um todo que é geralmente inseparável”
  - Essencialmente, abstrair é remover detalhes
  - Enquanto tenta-se manter a generalidade do raciocínio