Paradigmas de Programação

Prof. Vander Alves

O que caracteriza uma Linguagem de Programação?

- Gramática e significado bem definidos
- Implementável (executável) com eficiência "aceitável"
- Universal: deve ser possível expressar todo problema computável
- Natural para expressar problemas (em um certo domínio de aplicação)

Aspectos do estudo de linguagens

- Sintaxe: gramática (forma)
- Semântica: significado
- Pragmática (ex.: metodologias)
- Processadores:

  compiladores, interpretadores, editores, ambientes visuais ...

Por que tantas linguagens?

- Propósitos diferentes
- Avanços tecnológicos
- Interesses comercias
- Cultura e background científico

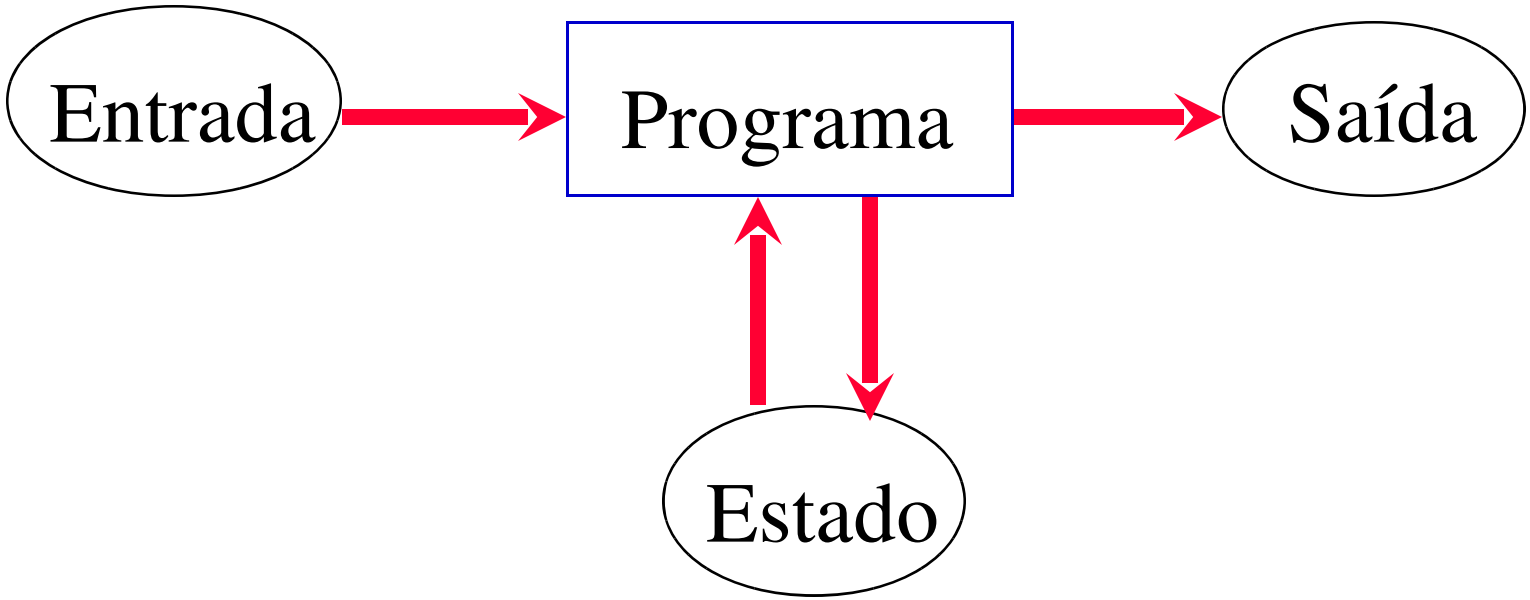
O que é um paradigma de programação?

- Modelo, padrão ou estilo de programação suportado por linguagens que agrupam certas características comuns
- A classificação de linguagens em paradigmas é uma conseqüência de decisões de projeto que impactam radicalmente a forma na qual uma aplicação real é modelada do ponto de vista computacional

O Paradigma Imperativo

- Programas centrados no conceito de um estado (modelado por variáveis) e ações (comandos) que manipulam o estado
- Paradigma também denominado de procedural, por incluir subrotinas ou procedimentos como mecanismo de estruturação
- Primeiro paradigma a surgir e ainda é o dominante

Modelo Computacional do Paradigma Imperativo



Vantagens do modelo imperativo

- Eficiência (embute modelo de Von Neumann)
- Modelagem "natural" de aplicações do mundo real
- Paradigma dominante e bem estabelecido

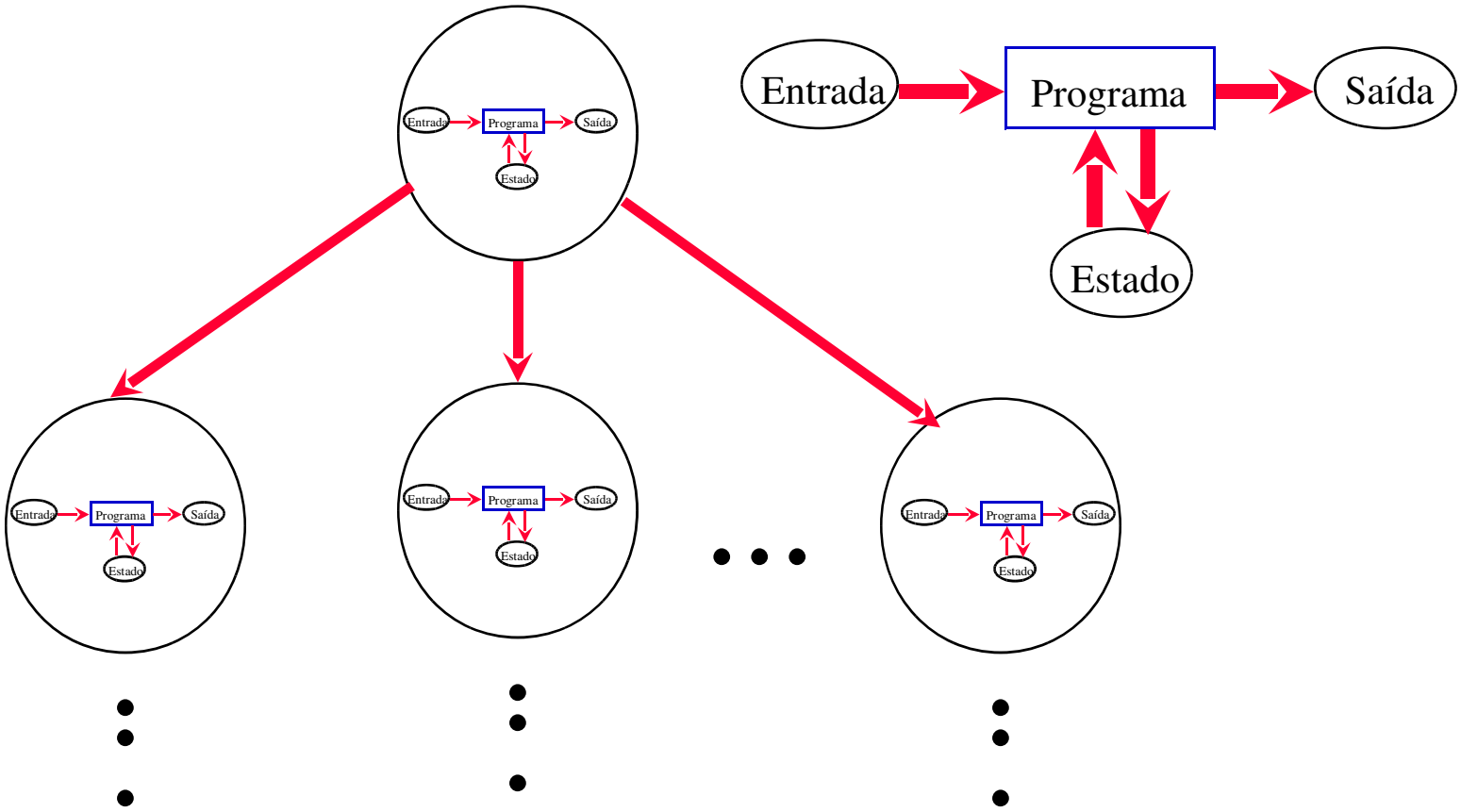
Desvantagens do paradigma imperativo

- Relacionamento indireto entre E/S resulta em:
  - difícil legibilidade
  - erros introduzidos durante manutenção
  - descrições demasiadamente operacionais
    focalizam o **como** e não **o que**

O Paradigma Orientado a Objetos

- Não é um paradigma no sentido estrito: é uma subclassificacão do imperativo
- A diferença é mais de metodologia quanto à concepção e modelagem do sistema
- A grosso modo, uma aplicação é estruturada em módulos (classes) que agrupam um estado (atributos) e operações (métodos) sobre este
- Classes podem ser estendidas e/ou usadas como tipos (cujos elementos são objetos)

Modelo Computacional do Paradigma Orientado a Objetos



Vantagens do Paradigma Orientado a objetos

- Todas as do estilo imperativo
- Classes estimulam projeto centrado em dados: modularidade, reusabilidade e extensibilidade
- Aceitação comercial crescente

Problemas do Paradigma OO

- Semelhantes aos do paradigma imperativo, mas amenizadas pelas facilidades de estruturação

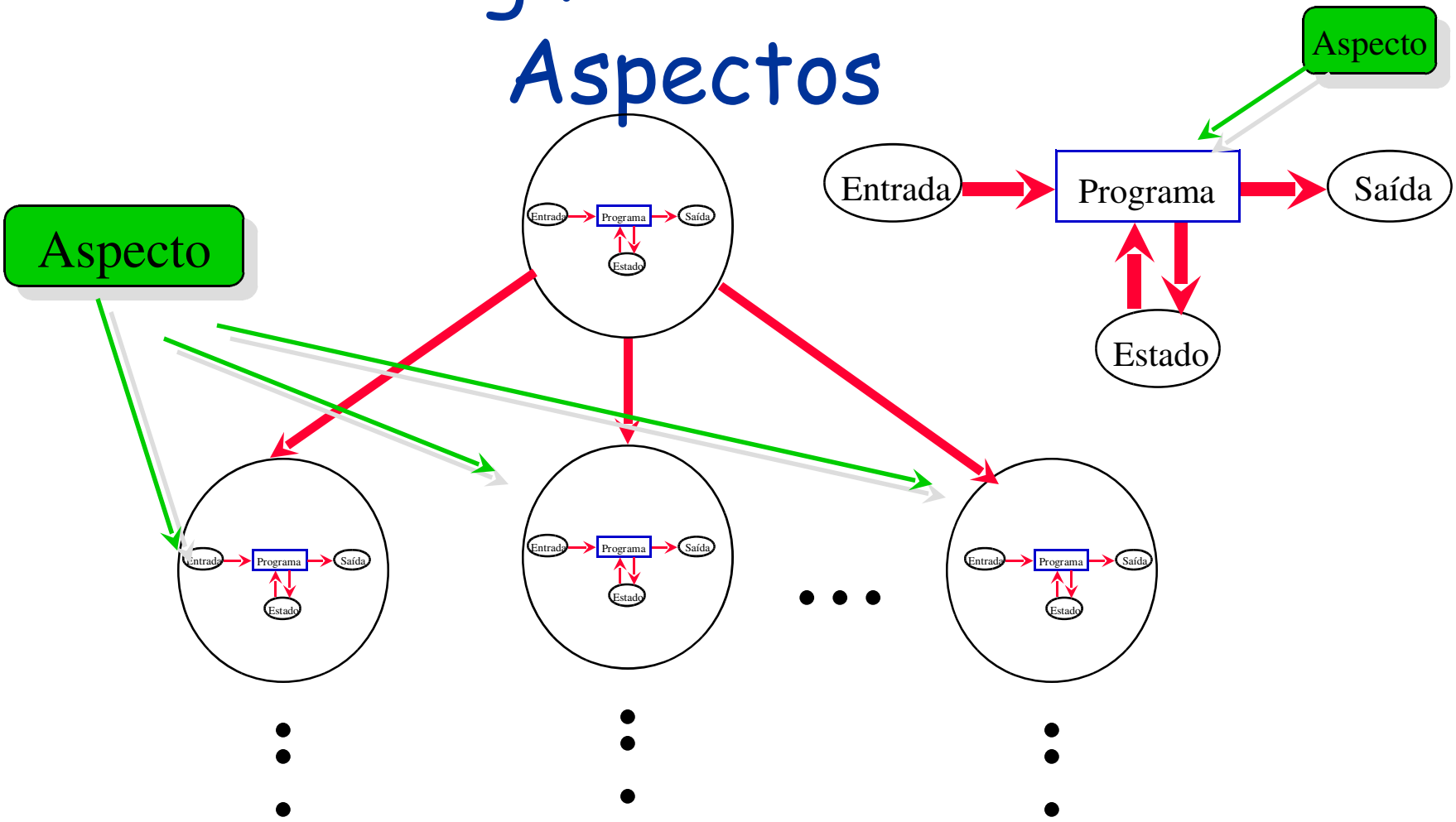
O Paradigma Orientado a Aspectos

- Não é um paradigma no sentido estrito
- A diferença é mais de metodologia quanto à concepção e modelagem do sistema
- É uma nova forma de modularização:
  - Para "requisitos" que afetam várias partes de uma aplicação

O Paradigma Orientado a Aspectos

- A grosso modo, uma aplicação é estruturada em módulos (aspectos) que agrupam pontos de interceptação de código (pointcuts) que afetam outros módulos (classes) ou outros aspectos, definindo novo comportamento (advice)
- Aspectos podem ser estendidos e/ou usados como tipos

Modelo Computacional do Paradigma Orientado a Aspectos



Vantagens do Paradigma Orientado a Aspectos

- Todas as do paradigma OO
- Útil para modularizar conceitos que a Orientação a Objetos não consegue (crosscutting concerns)
  - Em especial, aqueles ligados a requisitos não funcionais
- Aumenta a extensibilidade e o reuso
- Elevada expressibilidade e configurabilidade

Problemas do Paradigma Orientado a Aspectos

- Semelhantes aos do OO
- Ainda é preciso diminuir a relação entre classes e aspectos
- Problemas de conflito entre aspectos que afetam a mesma classe
- Necessidade de definir interfaces entre aspectos e objetos

O Paradigma Funcional

- Programas são funções que descrevem uma relação explícita e precisa entre E/S
- Estilo declarativo: não há o conceito de estado nem comandos como atribuição
- Conceitos sofisticados como polimorfismo, funções de alta ordem e avaliação sob demanda
- Aplicação: prototipação em geral e IA

Modelo Computacional do Paradigma Funcional

Visão Crítica do Paradigma Funcional

- Vantagens

  Manipulação de programas mais simples:

  - Prova de propriedades
  - Transformação (exemplo: otimização)
  - Concorrência explorada de forma natural

- Problemas

  "O mundo não é funcional!"

  Implementações ineficientes

  Mecanismos primitivos de E/S e formatação

O Paradigma Lógico

- Programas são relações entre E/S
- Estilo declarativo, como no paradigma funcional
- Na prática, inclui características imperativas, por questão de eficiência
- Aplicações: prototipação em geral, sistemas especialistas, banco de dados, ...

Modelo Computacional do Paradigma Lógico

Visão Crítica do Paradigma Lógico

- Vantagens

  Em princípio, todas do paradigma funcional

  Permite concepção da aplicação em um alto nível de abstração (através de associações entre E/S)

- Problemas

  Em princípio, todos do paradigma funcional

  Linguagens usualmente não possuem tipos, nem são de alta ordem

Outros "Paradigmas"

- Agentes
- ...

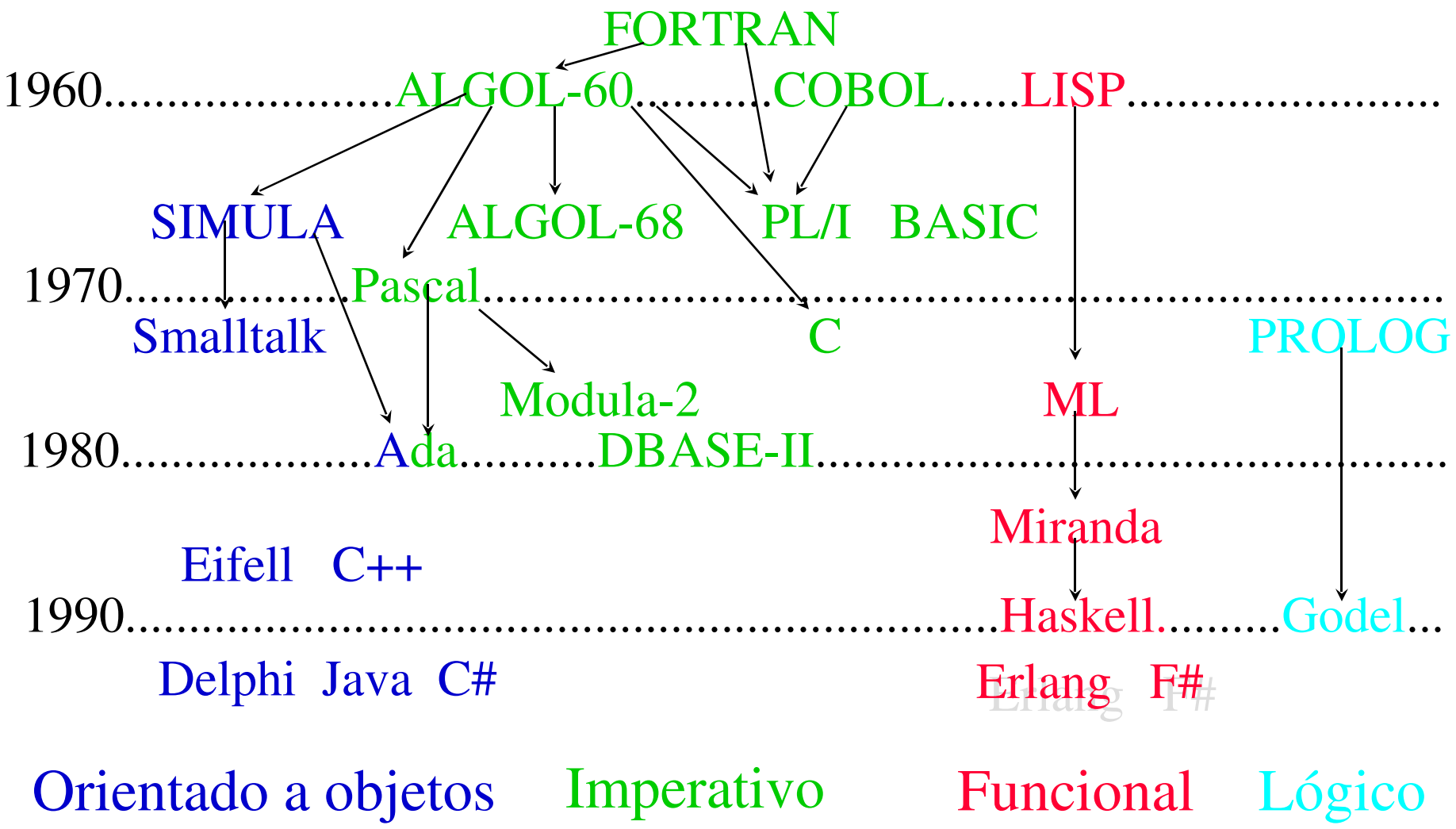
Tendência: integração de paradigmas

- A principal vantagem é combinar facilidades de mais de um paradigma, aumentando o domínio de aplicação da linguagem
- Exemplos: linguagens lógicas ou funcionais com o conceito de estado e comandos
- A integração deve ser conduzida com muita cautela, para que não se viole os princípios básicos de cada paradigma.

Outras Classificações

- Linguagens de 1a., 2a., 3a. 4a. e 5a. gerações
- Programação seqüencial versus concorrente
- Programação linear versus programação visual (*visual programming)*
- ...

Um breve histórico



Evolução centrada em níveis crescentes de abstração

- Linguagens de máquina
  - Endereços físicos e *operation code*
- Linguagens Assembly
  - Mnemônicos e labels simbólicos
- Linguagens de "alto nível"
  - Variáveis e atribuição (versus acesso direto à memória)
  - Estruturas de dados (versus estruturas de armazenamento)

Evolução centrada em níveis crescentes de abstração

- Estruturas de controle (versus *jumps* e *gotos*)
- Estrutura de blocos como forma de encapsulamento
- Generalização e parametrização (abstração de tipos de valores)