UFMG
UNIVERSIDADE FEDERAL
DE MINAS GERAIS

Programação e Desenvolvimento de Software 2

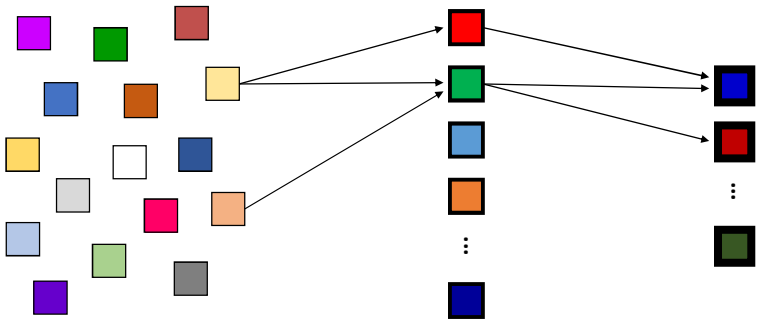
Introdução à Orientação a Objetos

Prof. Luiz Chaimowicz
(slides adaptados do Prof. Douglas Macharet)

DCC
DEPARTAMENTO DE
CIÊNCIA DA COMPUTAÇÃO

Introdução

Linguagens ➡ **Paradigmas** ➡ **Conceitos**



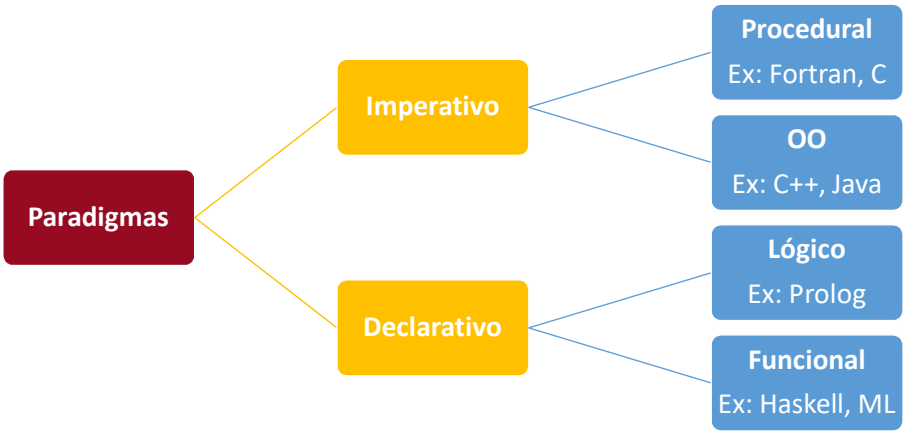
Cada linguagem realiza **um** ou **mais** paradigmas

Cada paradigma consiste em um conjunto de conceitos.

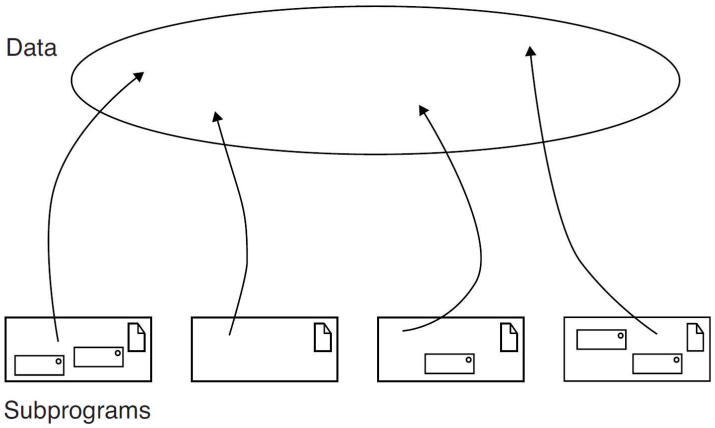
Introdução

- Paradigma de programação
  - Visão que o programador possui sobre a estruturação (organização do código) e execução do programa
  - Associado às técnicas de programação que permitem/proíbem
- Principais Tipos
  - Orientado a objetos ⬅
  - Estruturado (Procedural)
  - Funcional
  - Lógico

Introdução
Principais Paradigmas



Introdução
Programação Estruturada/Procedural

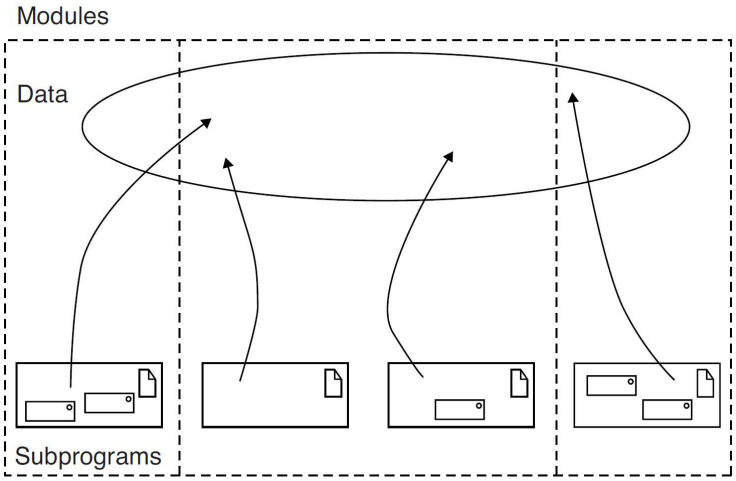


Fonte: *Object-Oriented Analysis and Design with Applications*

Introdução
Programação Estruturada/Procedural

- Como resolver problemas muito grandes?
  - Construí-lo a partir de partes menores
  - "Dividir para conquistar"
- Módulos compiláveis
  - Solucionam uma parte do problema
  - Dados x Manipulação
    - Abstração fraca para problemas mais complexos

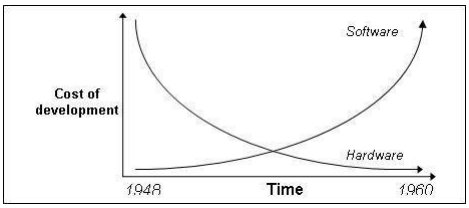
Introdução
Programação Estruturada/Procedural



Fonte: *Object-Oriented Analysis and Design with Applications*

Introdução
The Software Crisis

- Contexto (1960's)
  - Alta demanda por software
  - Sistemas maiores e mais complexos
  - Esforço de desenvolvimento cresceu
- Problemas
  - Prazos raramente cumpridos
  - Custos acima dos previstos
  - Não atendimento dos requisitos



http://www.chris-kimble.com/Courses/World_Med_MBA/Software_Crisis.html

https://en.wikipedia.org/wiki/Software_crisis
https://www.youtube.com/watch?v=0b5vp4Z2PKE
https://www.youtube.com/watch?v=Cd3TrUK8axU

Programação Orientada a Objetos

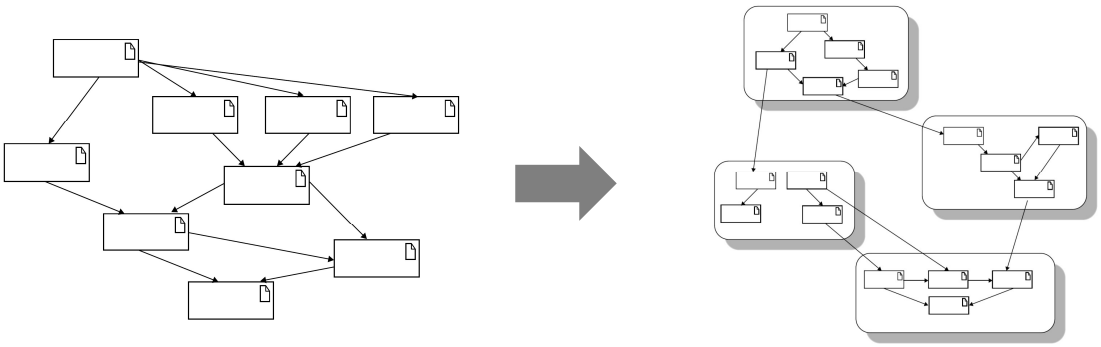
- Principais necessidades/objetivos
  - Aumentar a produtividade no desenvolvimento
  - Diminuir a chance de problemas na entrega
  - Facilitar a manutenção/extensão no futuro
- Programação Orientada a Objetos
  - Tem apresentado bons resultados
  - Não é uma bala de prata!

https://en.wikipedia.org/wiki/Object-oriented_programming#Criticism

Programação Orientada a Objetos

- Desenvolvimento de hardware
  - Inspiração → Coleção de pedaços simples de hardware (chips) unidos para se montar um hardware mais complexo
- Amadurecimento dos conceitos
  - Simula (1960's)
  - Smalltalk (1970's)
  - C++ (1980's)
  - Java (1990's)

Programação Orientada a Objetos



Fonte: *Object-Oriented Analysis and Design with Applications*

Programação Orientada a Objetos
PE vs. POO

- Programação Estruturada
  - Procedimentos implementados em blocos
  - Comunicação pela passagem de dados
  - Execução → Acionamento de procedimentos
- Programação Orientada a Objetos
  - Dados e procedimentos encapsulados (TADs)
  - Execução → "Comunicação" entre objetos

Programação Orientada a Objetos
PE vs. POO

- Programação Estruturada
  - Dados globais são acessados via funções (estado)
  - Representação de tipos complexos com *struct/typedef*

- Programação Orientada a Objetos
  - Dados (tipos) são dotados de certa inteligência
  - Conhecem e sabem realizar operações sobre si mesmos
  - Preciso saber como fazem isso?
    - Não! ➔ Especificação/Implementação

O que o TAD representa e faz é mais importante do que como ele faz!

Programação Orientada a Objetos
"Humanização" dos dados

- Os dados/tipos têm "características humanas"
  - Conhecimento/ação sobre si próprios!

- Exemplos
  - Uma *circunferência* sabe determinar sua área
  - Uma *lista* sabe dizer quantos elementos tem
  - Um *estudante* sabe se matricular em um *curso*

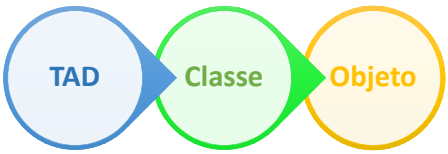
Classes

- Classe
  - Estruturada em uma unidade de compilação
    - Define uma lógica estática (trecho de código)
  - Representação → Conceito, ideia, abstração
  - Struct turbinado! 💪
- Formaliza como compreendemos algo no domínio do problema
- Conjunto de elementos com propriedades/operações semelhantes

Mundo Real

Objetos

- Objeto é uma **instância** de uma Classe
  - Representa um elemento específico do conjunto
- Existe em tempo de execução (está na memória)
- TAD é implementado por uma Classe que gera Objetos



Classes vs. Objetos

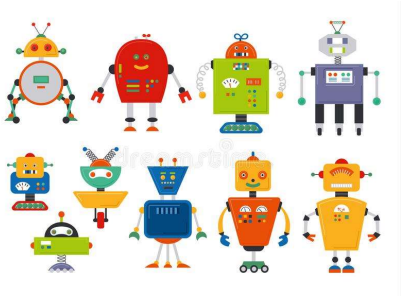
CLASSE

OBJETOS

Robot

Atributos:
cor, tipo, #rodas/pernas, …

Métodos:
mover, pegar, …



Classes vs. Objetos

Massa	Fôrma	Biscoitos
Memória	Classes	Objetos

Objetos

- Possuem alocação própria na memória
  - Devem ser criados (instanciados) explicitamente
  - Em algum momento serão destruídos (Explícito x Implícito)
- Propriedades

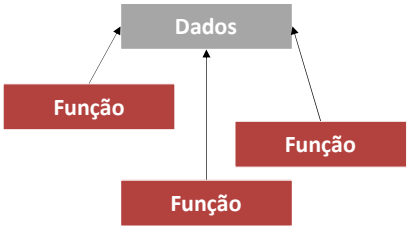
Estado | **Comportamento** | **Identidade**

Objetos

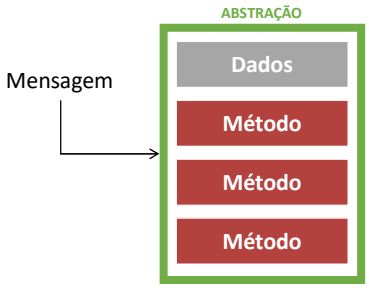
- Estado
  - Valores que os atributos possuem em um certo momento
- Comportamento
  - Responsável pelas mudanças de estado
  - Relacionamento com os demais objetos
- Identidade (referência única)
  - Propriedade que o difere dos demais objetos na memória

Objetos

- Dados ocultos do "mundo externo" (Distribuídos ≠ Espalhados)
- Devem ser acessíveis somente via métodos internos

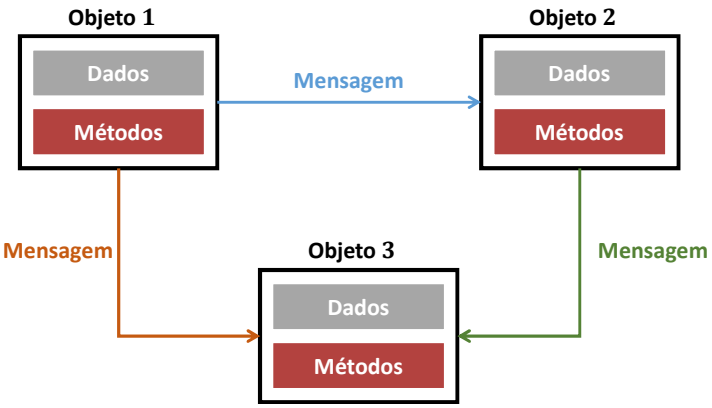


Programação Estruturada



Programação Orientada a Objetos

Objetos



"I was too blythe about the term [objected oriented] back in the 60s and should have chosen something like *message oriented*".

– Alan Kay, Jun 8, 2011 at 16:27

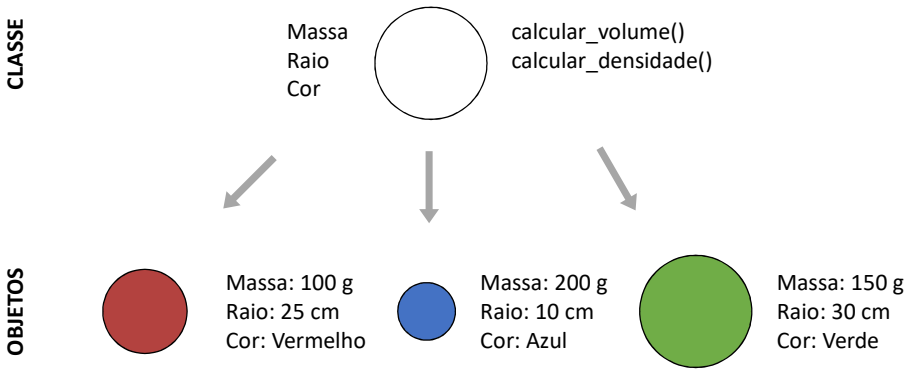
"Alan Kay has argued that message passing is more important than objects in OOP, and that objects themselves are often over-emphasized."

https://softwareengineering.stackexchange.com/questions/46592/so-what-did-alan-kay-really-mean-by-the-term-object-oriented
https://en.wikipedia.org/wiki/Message_passing

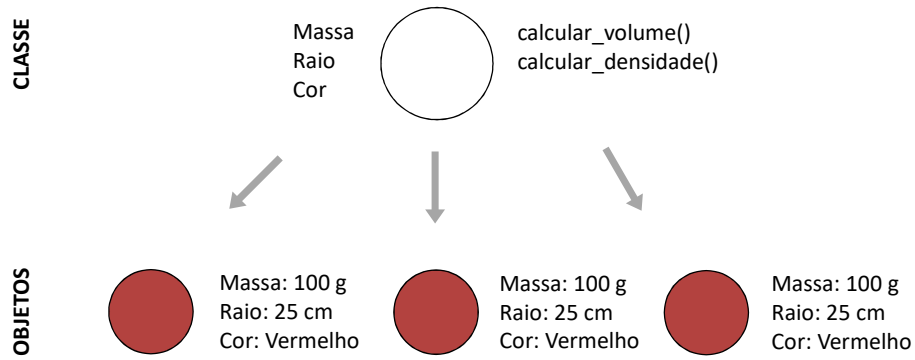
Classes vs. Objetos

- Classe
  - Descrição de propriedades em comum de um grupo de objetos (**conjunto**)
  - Um **conceito**
  - Faz parte de um **programa**
  - Exemplo: Pessoa
  - Exemplo: Carro
- Objeto
  - Representação das propriedades de uma única instância (**elemento**)
  - Um **fenômeno** (ocorrência)
  - Faz parte de uma **execução**
  - Exemplo: João, Maria
  - Exemplo: Ferrari, Fusca

Classes vs. Objetos



Classes vs. Objetos



Exemplo

Isso são **modificadores de acesso** e serão detalhados nas próximas aulas!

Esfera.hpp

```
#ifndef ESFERA_H
#define ESFERA_H

#include <string>
class Esfera {
  private:
    float massa;
    float raio;
    std::string cor;
 public:
    Esfera(float massa,float raio,
           std::string cor);
    float calcular_volume();
    float calcular_densidade();
};

#endif
```

Esfera.cpp

```
#include <cmath>
#include "Esfera.hpp"

Esfera::Esfera(float massa, float raio,
               std::string cor){
    this->massa = massa;
    this->raio = raio;
    this->cor = cor;
}

float Esfera::calcular_volume() {
   return (4/3)*M_PI*pow(this->raio, 3);
}

float Esfera::calcular_densidade() {
   return this->massa/this->calcular_volume();
}
```

Exemplo

main.cpp

```
#include <iostream>
#include "Esfera.hpp"

using namespace std;

int main() {
  Esfera esf1(100, 25, "vermelho");
  cout << "Densidade Esf1: " << esf1.calcular_densidade() << endl;

  Esfera *esf2 = new Esfera(100, 25, "vermelho");
  cout << "Densidade Esf2: " << esf2->calcular_densidade() << endl;

  delete esf2;

  return 0;
}
```


Programação Orientada a Objetos
Princípios

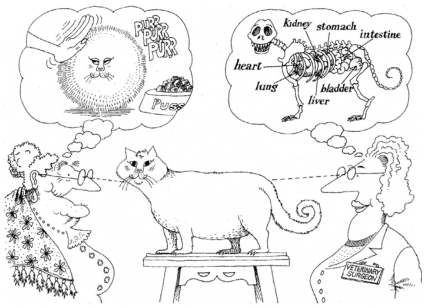
- Abstração
- Encapsulamento
- Herança
- Polimorfismo
- Mensagens
- Modularidade

Princípios fundamentais

Programação Orientada a Objetos
Princípios – Abstração

- Modelagem de um domínio
  - Identificar artefatos de software
  - Ignorar aspectos não relevantes
  - Representação de detalhes do domínio do problema na linguagem de solução
- Classes são abstrações de conceitos

"Concentrar-se nas características essenciais de algum elemento, em relação à perspectiva do observador."



Fonte: *Object-Oriented Analysis and Design with Applications*

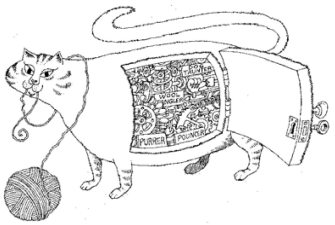
Programação Orientada a Objetos
Princípios – Abstração

Abstração
- Elimine o irrelevante, enfatize o essencial.

Programação Orientada a Objetos
Princípios – Encapsulamento

- Agrupamento dos dados e procedimentos correlacionados em uma mesma entidade
- Sistema orientado a objetos baseia-se no contrato e não na implementação interna
- Proteção da estrutura interna (integridade)

"Esconder os detalhes da implementação de um objeto."



Fonte: *Object-Oriented Analysis and Design with Applications*

Programação Orientada a Objetos
Princípios – Encapsulamento

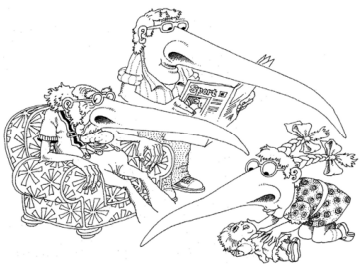
Abstração
- Elimine o irrelevante, enfatize o essencial.

Encapsulamento
- Exiba apenas o necessário, esconda o resto.

Programação Orientada a Objetos
Princípios – Herança

- Permite a hierarquização das classes
- Classe especializada (subclasse, filha)
  - Herda as propriedades (atributos e métodos)
  - Pode sobrescrever/estender o comportamento
- Auxilia no reuso de código

"As subclasses podem herdar a estrutura e comportamento de sua superclasse."



Fonte: *Object-Oriented Analysis and Design with Applications*

Programação Orientada a Objetos
Princípios – Herança

Abstração
- Elimine o irrelevante, enfatize o essencial.

Encapsulamento
- Exiba apenas o necessário, esconda o resto.

Herança
- Modele a semelhança, mas permita a diferença.

Programação Orientada a Objetos
Princípios – Polimorfismo

- Tratar tipos diferentes de forma homogênea
  - Classes distintas com métodos homônimos
  - Diferentes níveis na mesma hierarquia
- Um método assume "diferentes formas"
  - Apresenta diferentes comportamentos

"Invocar um mesmo comportamento de objetos diferentes e obter respostas específicas."

Programação Orientada a Objetos
Princípios – Polimorfismo

Abstração
- Elimine o irrelevante, enfatize o essencial.

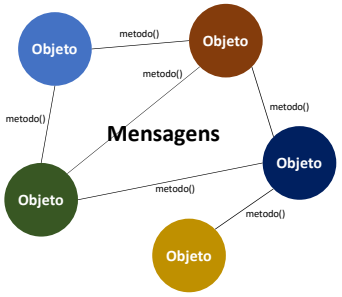
Encapsulamento
- Exiba apenas o necessário, esconda o resto.

Herança
- Modele a semelhança, mas permita a diferença.

Polimorfismo
- Mesmo contrato, comportamentos diferentes.

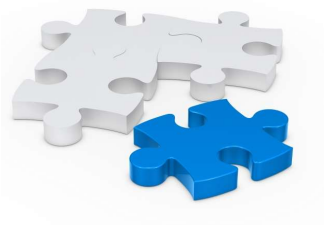
Programação Orientada a Objetos
Princípios – Mensagens

- Interação entre objetos
  - Envio/recebimento de *mensagens*
  - Invocar um comportamento específico
- Informação contida na mensagem
  - Utiliza o contrato firmado entre as partes
  - Parâmetros de entrada e saída



Programação Orientada a Objetos
Princípios – Modularidade

- Separação em conjuntos de módulos
  - Classes com independência de funcionamento
- Separação de responsabilidades
  - Limites lógicos entre os componentes
  - Melhora a manutenibilidade
  - Aumenta a estabilidade (↓ efeitos colaterais)



Considerações finais

- Maior confiabilidade
- Maior reaproveitamento de código
- Facilidade de manutenção
- Melhor gerenciamento
- Maior robustez
- ...