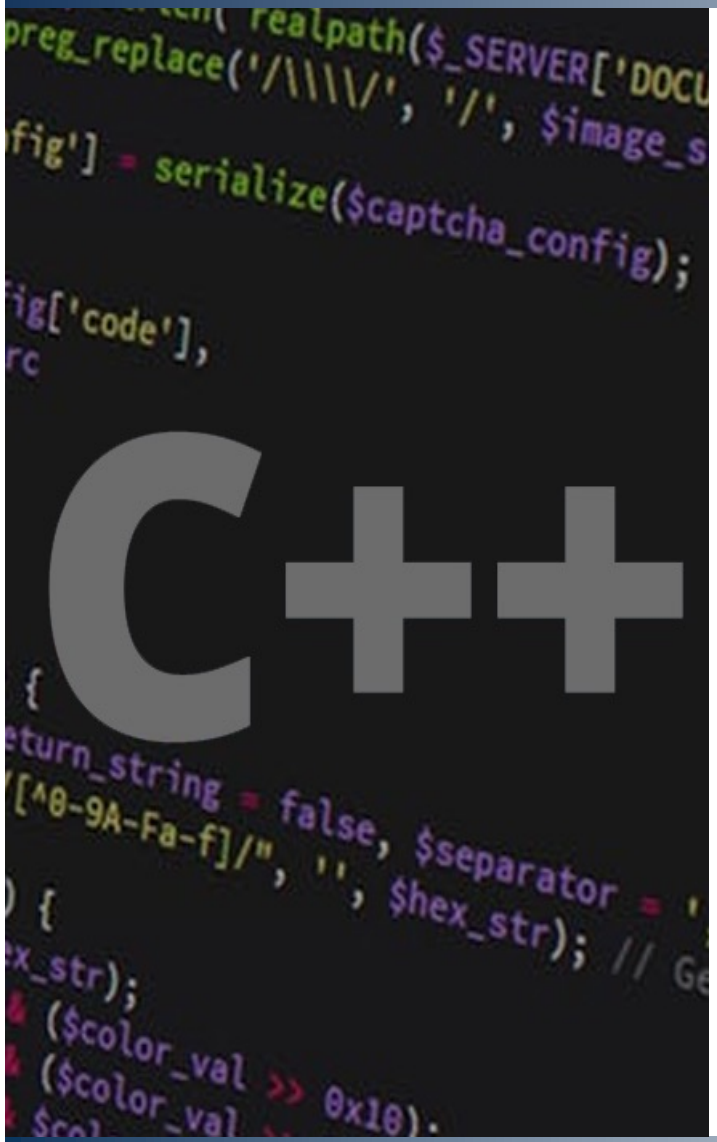
UFAC



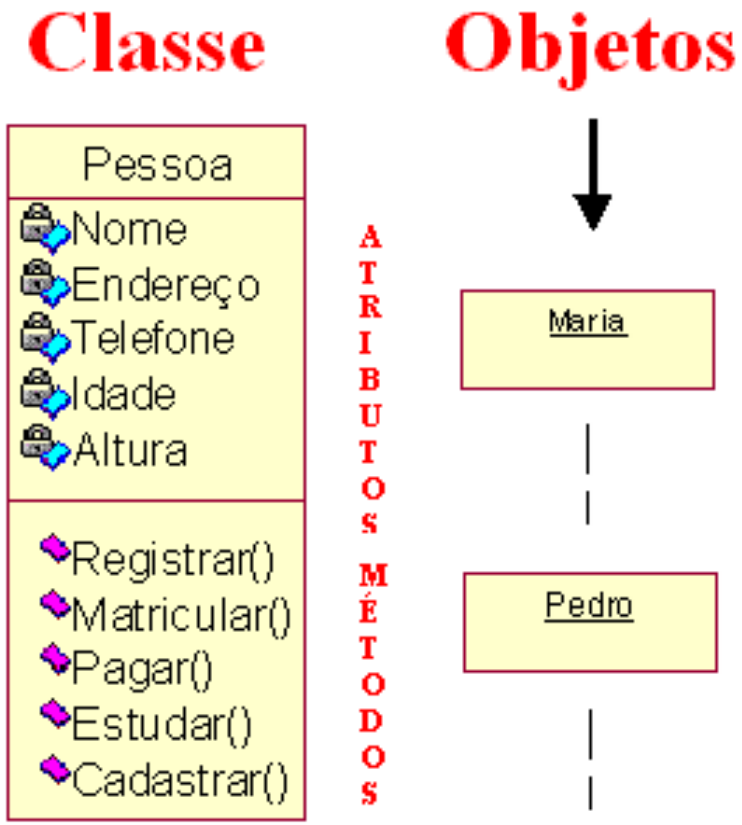
Linguagem de Programação II

Prof. André Nasserala

andre.nasserala@ufac.br

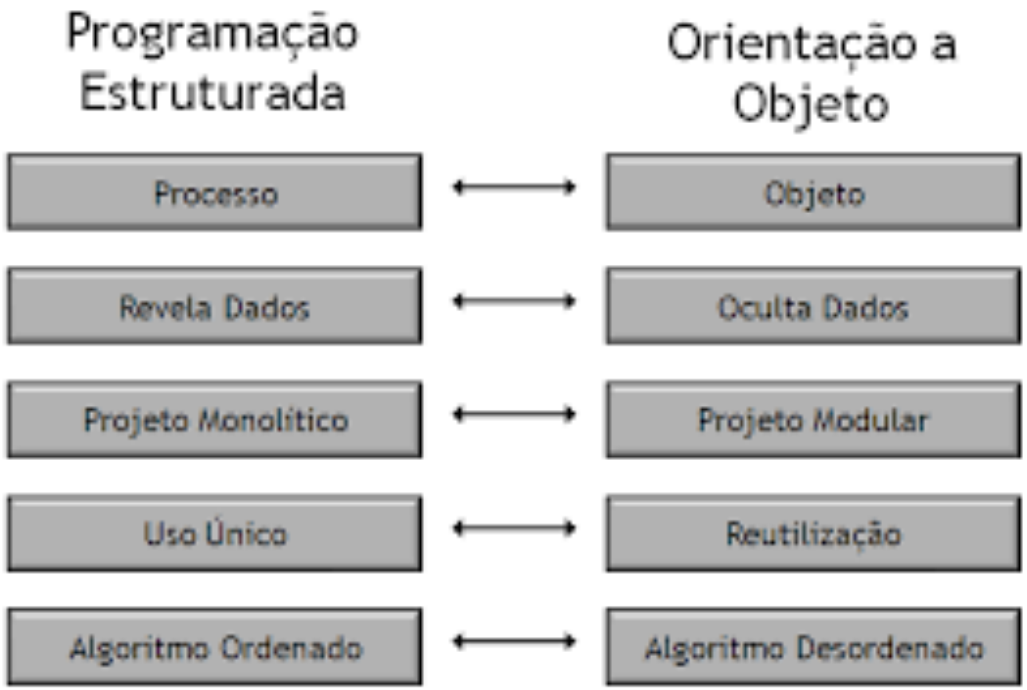
Orientação a Objetos

- A Orientação a Objetos (POO) é um paradigma de programação que organiza o código em torno de "objetos", unidades básicas que encapsulam dados ("atributos") e comportamentos ("métodos").
- Inspirada no mundo real, a POO visa criar programas mais flexíveis, reutilizáveis e fáceis de manter, especialmente para projetos complexos.



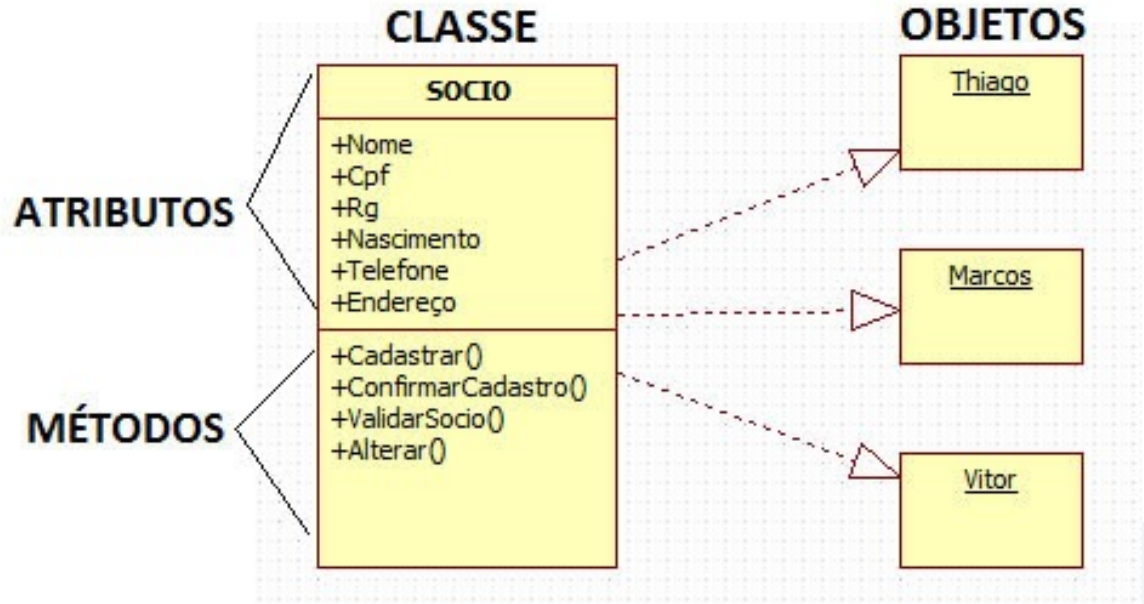
Orientação a Objetos

- A POO é uma ferramenta poderosa para a criação de software robusto, flexível e manuseável.
- Dominá-la permite aos programadores construir soluções eficientes e escaláveis para diversos desafios.



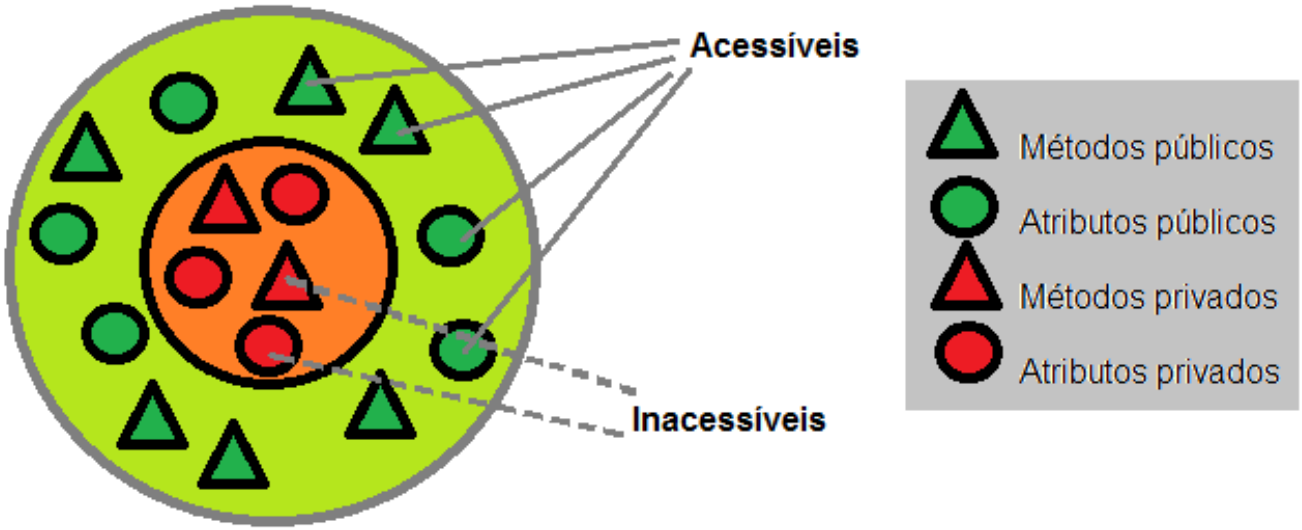
Conceitos Fundamentais da POO

- **Objetos**: Representam entidades do mundo real ou conceitos abstratos. Possuem:
  - Atributos: Armazenam dados que definem o estado do objeto.
  - Métodos: Definem as ações que o objeto pode realizar.
- **Classes**: Modelos que definem a estrutura dos objetos, especificando seus atributos e métodos.
  - Funcionam como "planos" para a criação de objetos.



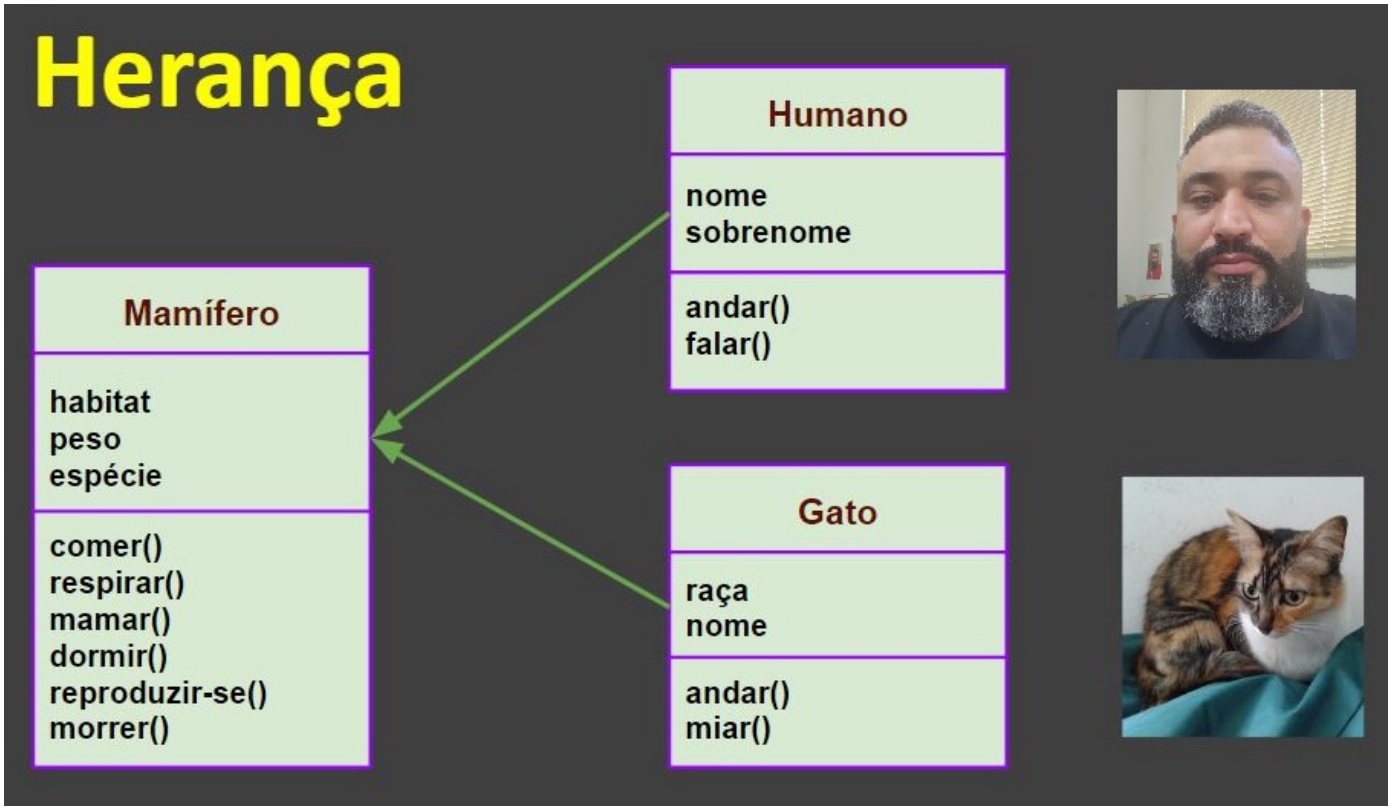
Conceitos Fundamentais da POO

- **Encapsulamento**: Esconde a implementação interna dos objetos, expondo apenas interfaces públicas (métodos) para interação. Promove segurança e modularidade.



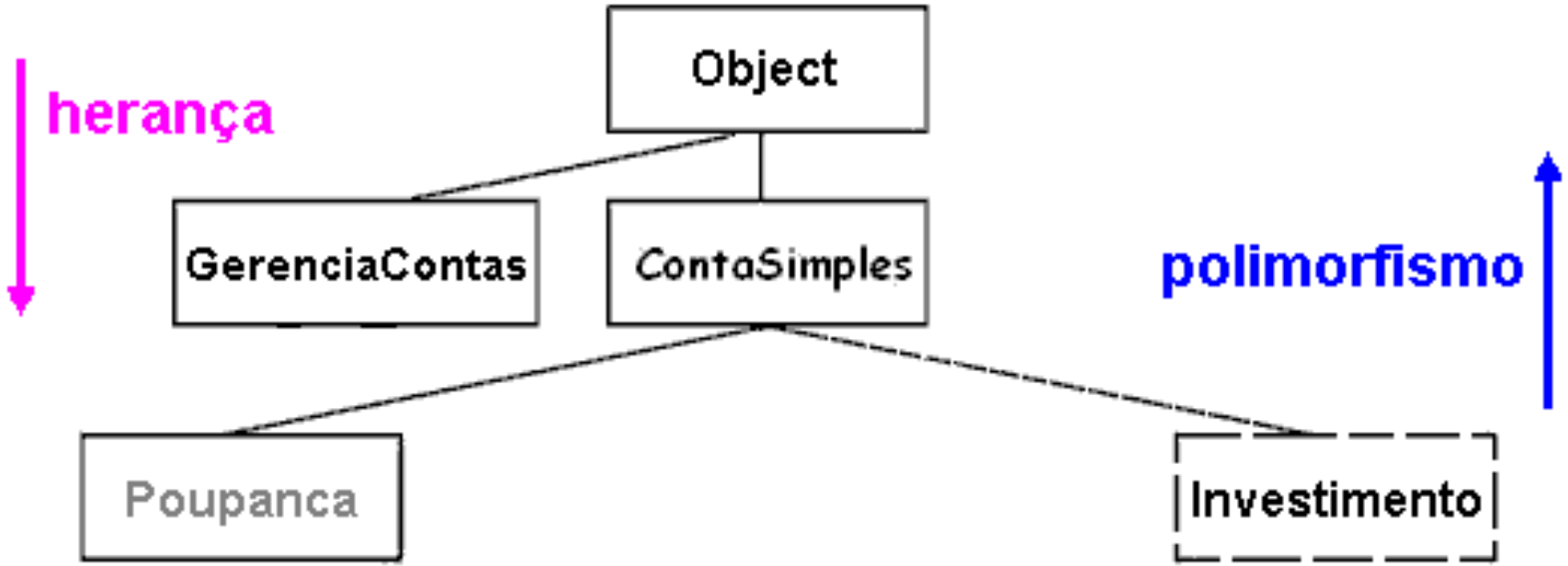
Conceitos Fundamentais da POO

- **Herança**: Permite que classes herdem atributos e métodos de outras classes, promovendo reuso de código e organização hierárquica.



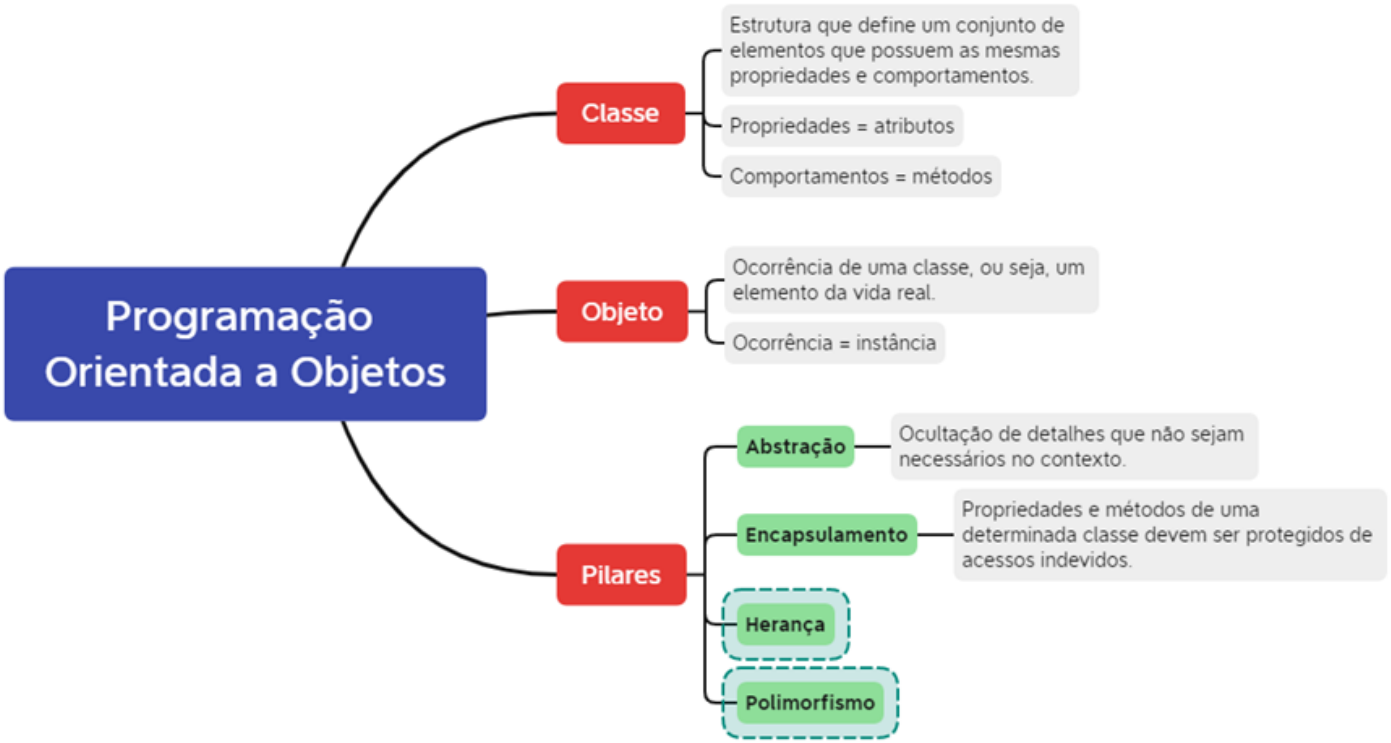
Conceitos Fundamentais da POO

- **Polimorfismo**: Permite que objetos de diferentes classes respondam à mesma mensagem de maneiras distintas, proporcionando flexibilidade e abstração.



Conceitos Fundamentais da POO

- **Abstração**: Foca nas características essenciais de um objeto, ignorando detalhes desnecessários, simplificando o uso e a compreensão do código.



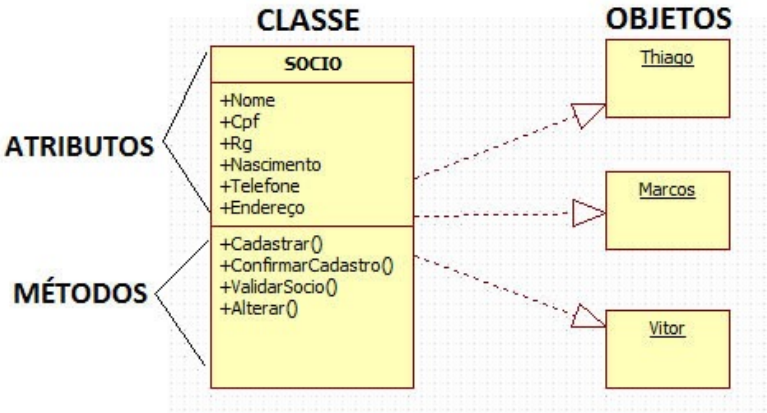
Benefícios da POO

- Modularidade: Divisão do programa em módulos independentes e reutilizáveis.
- Reusabilidade: Código escrito em uma classe pode ser usado por outras classes, reduzindo duplicação e aumentando a produtividade.
- Manutenabilidade: Facilidade de modificar e atualizar o código, pois as alterações se concentram em classes específicas.
- Flexibilidade: Adaptabilidade a mudanças nos requisitos do projeto.
- Legibilidade: Código mais organizado e expressivo, facilitando a compreensão e colaboração.

Classes e Objetos

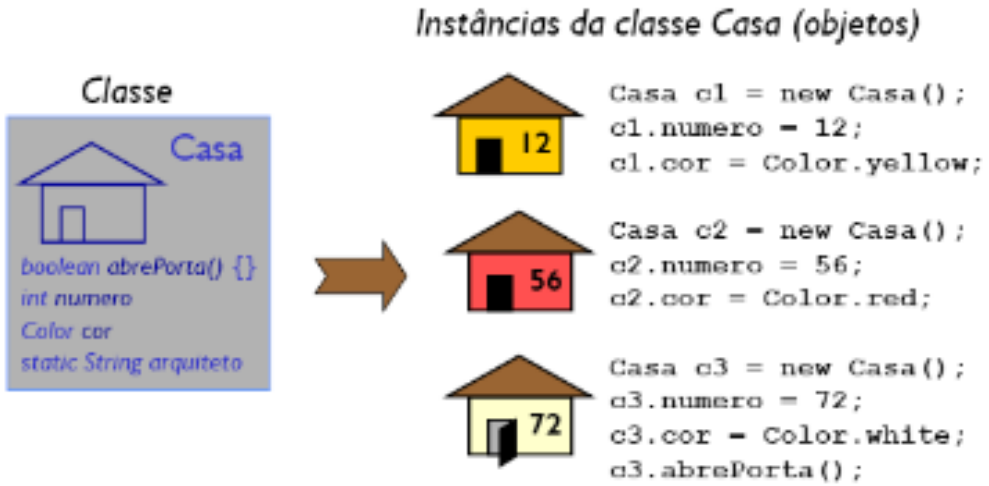
- A ideia fundamental de linguagens orientadas ao objeto é a possibilidade de combinar num único registro campos que conterão dados e campos que são funções para operar os campos de dados do registro.
- Uma unidade assim definida é chamada classe.
- Uma classe é considerada um tipo de dado como os tipos que existem predefinidos em compiladores de diversas linguagens de programação.
- Como exemplo, considere o tipo int que é predefinido em C e C++.

```
class Pessoa
{
private:
    char nome[100];
    int idade;
    float peso;
};
```



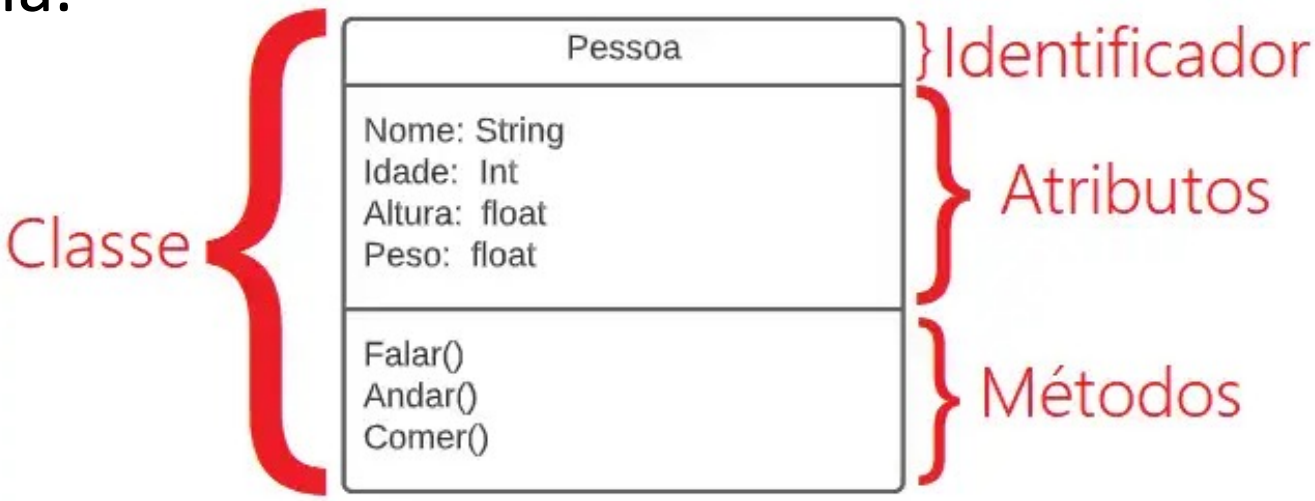
Classes e Objetos

- Podemos declarar quantas variáveis do tipo int forem necessárias ao programa.
- De modo similar, podemos declarar quantas variáveis quisermos de uma classe já definida.
- Uma variável de uma classe é chamada objeto e conterá campos de dados e funções.



Classes e Objetos

- Definir uma classe não cria nenhum objeto, do mesmo modo que a existência do tipo int não cria nenhuma variável.
- As funções de um objeto são chamadas funções-membro ou métodos e, de modo geral, são o único meio de acesso aos campos de dados também chamados variáveis de instância.



Encapsular e Esconder

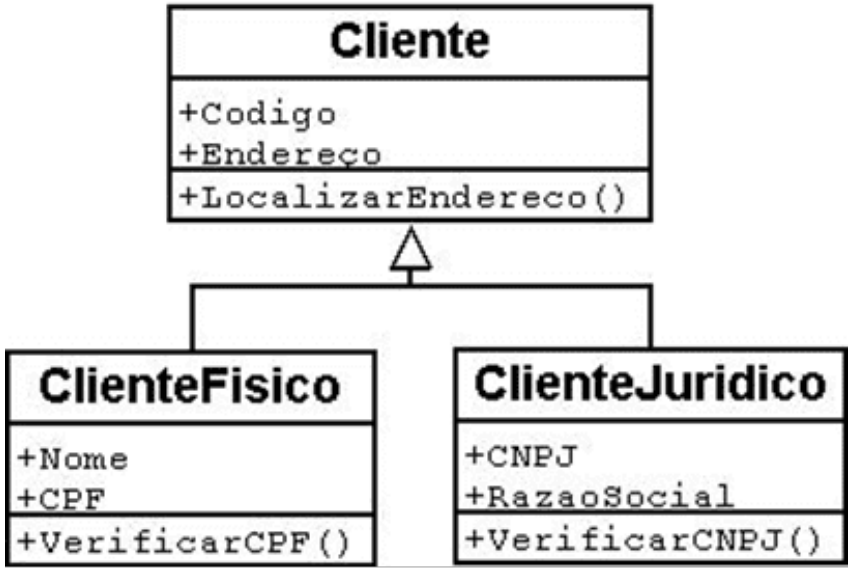
- Se o programa necessita atribuir um valor a alguma variável de instância, deve chamar uma função-membro que recebe o valor como argumento e faz a alteração.
- Não podemos acessar variáveis de instância diretamente.
- Desta forma, os campos de dados estarão escondidos para nós, o que previne alterações acidentais.
- Dizemos então que os campos de dados e suas funções estão encapsulados (de cápsula) numa única entidade.
- As palavras encapsular e esconder são termos técnicos da definição de linguagens orientadas ao objeto.

Encapsular e Esconder

- Se alguma modificação ocorrer em variáveis de instância de um certo objeto, sabemos exatamente quais funções interagiram com elas: são as funções-membro do objeto.
- Nenhuma outra função pode acessar esses dados.
- Isso simplifica a escrita, manutenção e alteração de programas.
- Um programa em C++ consiste em um conjunto de objetos que se comunicam por meio de chamadas às funções-membro.
- A frase "chamar uma função-membro de um objeto" pode ser dita como "enviar uma mensagem a um objeto".

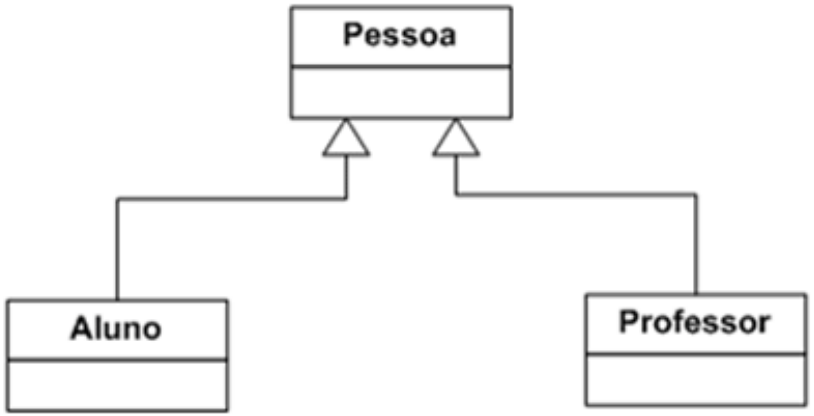
Herança

- A programação orientada ao objeto oferece uma maneira de relacionar classes umas com as outras por meio de hierarquias.



Herança

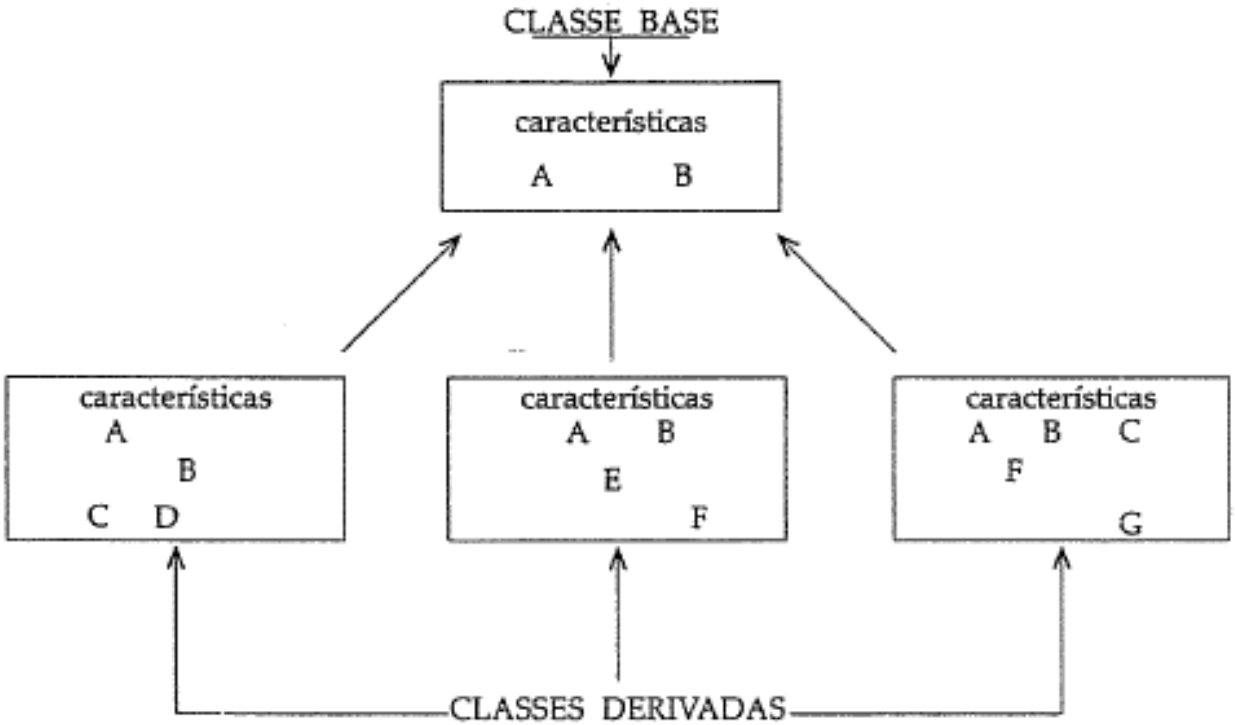
- No nosso dia-a-dia, esse processo está presente quando dividimos classes em subclasses, mantendo-se o princípio de que cada subclasse herda as características da classe da qual foi derivada.
- Por exemplo: a classe de animais é dividida nas subclasses mamíferos, aves, peixes etc.
- Uma das características da classe animais é a reprodução.



Herança: Aluno e Professor herdam de Pessoa

Herança

- Todas as suas subclasses têm essa característica. Além das características herdadas, cada subclasse tem suas características particulares.



Herança

- Em programação orientada ao objeto, o conceito de subclasse ou processo de classes derivadas é chamado HERANÇA.
- Em C++, a classe de origem é chamada classe-base e as classes que compartilham as características de uma classe-base e têm outras características adicionais são chamadas classes derivadas.
- Uma classe-base representa os elementos comuns a um grupo de classes derivadas.

Herança

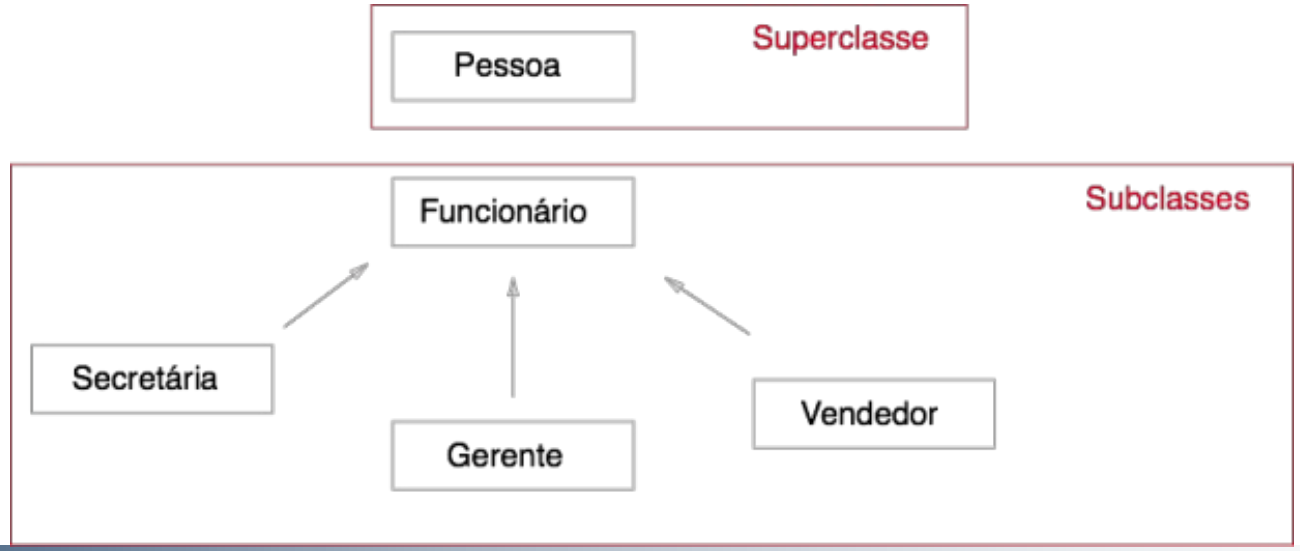
- Você poderia pensar em herança como algo semelhante ao uso de funções para simplificar tarefas tradicionais.
- Você escreve uma função quando identifica que várias secções diferentes de um programa, em parte, executam a mesma coisa.
- Em C++, você define uma classe-base quando identifica características comuns em um grupo de classes derivadas.
- Da mesma forma que você pode criar uma biblioteca de funções úteis a diversos programas, pode formar uma biblioteca de classes que poderão vir a ser o núcleo de muitos programas.

Herança

- O uso de uma biblioteca de classes oferece uma grande vantagem sobre o uso de uma biblioteca de funções: o programador pode criar classes derivadas de classes-base de biblioteca.
- Isto significa que, sem alterar a classe-base, é possível adicionar a ela características diferentes que a tornarão capaz de executar exatamente o que desejarmos.
- Uma classe que é derivada de uma classe-base pode, por sua vez, ser a classe-base de outra classe.
- O uso de classes derivadas aumenta a eficiência da programação pela não-necessidade da criação de códigos repetitivos.

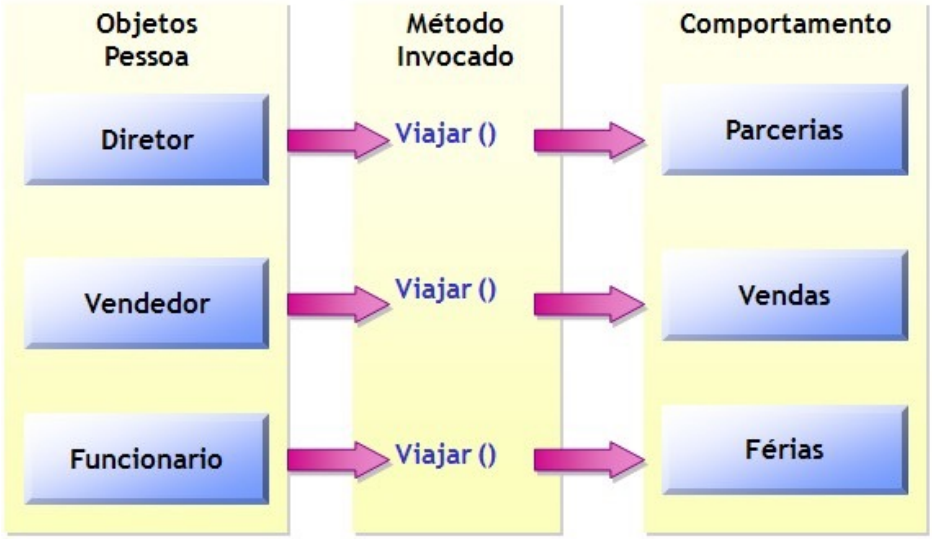
Herança

- A uma função de biblioteca não podemos adicionar outras implementações a não ser que ela seja reescrita ou que tenhamos seu código-fonte para alterá-la e recompilá-la.
- A facilidade com que classes existentes podem ser reutilizadas sem serem alteradas é um dos maiores benefícios oferecidos por linguagens orientadas ao objeto.



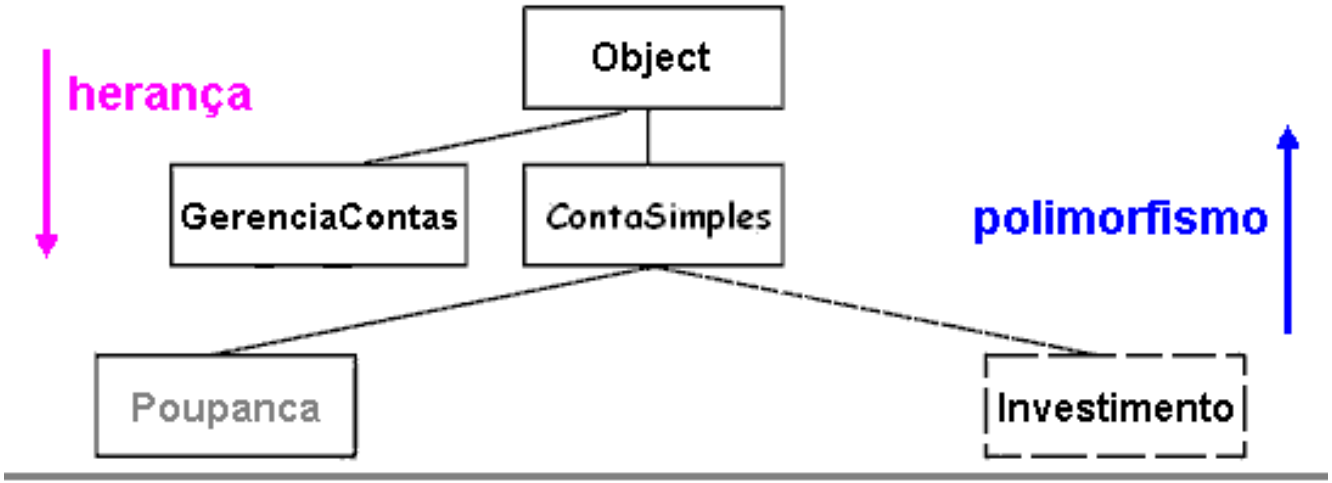
Poliformismo e Sobrecarga

- O sentido da palavra polimorfismo é o do uso de um único nome para definir várias formas distintas.
- Em C++, chamamos de polimorfismo a criação de uma família de funções que compartilham do mesmo nome, mas cada uma tem código independente.
- O resultado da ação de cada uma das funções da família é o mesmo.
- A maneira de atingir esse resultado é distinta.



Poliformismo e Sobrecarga

- Como exemplo, suponhamos que desejemos calcular o salário líquido de todos os funcionários de uma empresa.
- Nessa empresa há horistas, mensalistas, os que ganham por comissão etc.
- Criamos um conjunto de funções em que cada uma tem um método diferente de calcular o salário.
- Quando a função é chamada, C++ saberá identificar qual é a função com o código adequado ao contexto.



Poliformismo e Sobrecarga

- A sobrecarga é um tipo particular de polimorfismo.
- Como exemplo, tomemos o operador aritmético +.
- Em C++, usamos esse operador para somar números inteiros ou para somar números reais:
- 5 + 2
- 3 . 4 + 5 . 8
- O computador executa operações completamente diferentes para somar números inteiros e números reais.
- A utilização de um mesmo símbolo para a execução de operações distintas é chamada sobrecarga.

Poliformismo e Sobrecarga

- Em C++, além das sobrecargas embutidas na linguagem, podemos definir outras com capacidades novas.
- Por exemplo, podemos definir a seguinte operação nova para o símbolo +:

- “ABC“ + “DEF” = “ABCDEF”

- O polimorfismo e a sobrecarga são habilidades únicas em linguagens orientadas ao objeto.

C é um Subconjunto de C++

- C++ é uma linguagem derivada da linguagem C.
- O conjunto de instruções que fazem parte da linguagem C também é parte de C++.
- Os elementos principais que foram adicionados à linguagem C para dar origem a C++ consistem nas classes, nos objetos e na ideia de programação orientada ao objeto.
- Se você já sabe programar em C, conhece a maior parte da sintaxe de C++ e tem pouco a aprender.
- C++ é rica em recursos que atendem às limitações impostas pelas linguagens procedurais.

Bibliografia

- MIZRAHI1, Victorine Viviane. Treinamento Em Linguagem C ++ - Modulo 1. 2ª Edição. Editora: Pearson Universidades. Ano: 2005. ISBN: 8576050455.
- VIEIRA, Lucas S. Introdução a Programação em C++. Diamantina - MG. Creative Commons – 2019.
- PEREIRA, Silvio do Lago. Linguagem C++. São Paulo: FATEC, 1999.
- MIZRAHI2, Victorine Viviane. Treinamento Em Linguagem C ++ - Modulo 2. 2ª Edição. Editora: Pearson Universidades. Ano: 2005. ISBN: 8576050463.