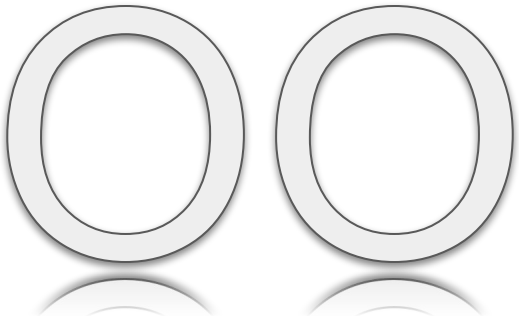
POO: Herança

Eng. Computação

Prof. Victor A P Oliveira

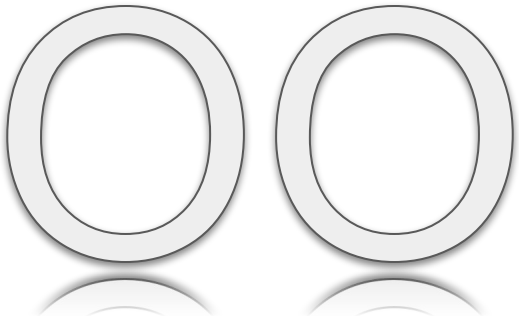
Introdução



Pilares:

- Abstração
- Encapsulamento
- Herança
- Polimorfismo

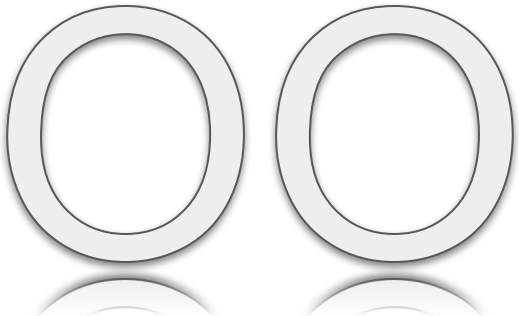
Introdução



Pilares:

- Abstração 🍏
- Encapsulamento
- Herança
- Polimorfismo

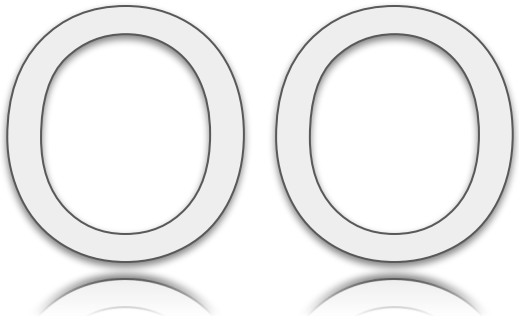
Introdução



Pilares:

- Abstração
- Encapsulamento
- Herança
- Polimorfismo

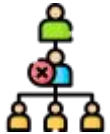
Introdução



Pilares:

- Abstração
- Encapsulamento
- Herança
- Polimorfismo

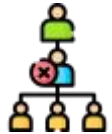
Conceitos Fundamentais



Herança

é uma forma de **reutilização de software** em que o programador cria uma classe que ***absorve*** atributos e comportamentos de uma classe existente e os aprimora com *novas capacidades* (Deitel)

Conceitos Fundamentais



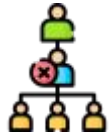
Herança

é uma forma de **reutilização de software** em que o programador cria uma classe que ***absorve*** atributos e comportamentos de uma classe existente e os aprimora com *novas capacidades* (Deitel)

Benefícios:

- **reusabilidade** de software já testado
- sistema mais **enxuto e eficiente**
- ganho de **produtividade**
- para que reinventar a roda?

Conceitos Fundamentais

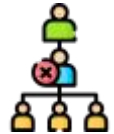


Classe básica x Classe derivada

Classe básica é a Classe já existente, a Classe que será herdada. Também conhecida por **superclasse**, **classe mãe** **...**

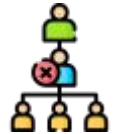
Classe derivada é a nova Classe, a que aproveita os membros (atributos e métodos) da Classe básica. Também chamada de **subclasse**, **classe filha**, **especialização** etc.

Conceitos Fundamentais

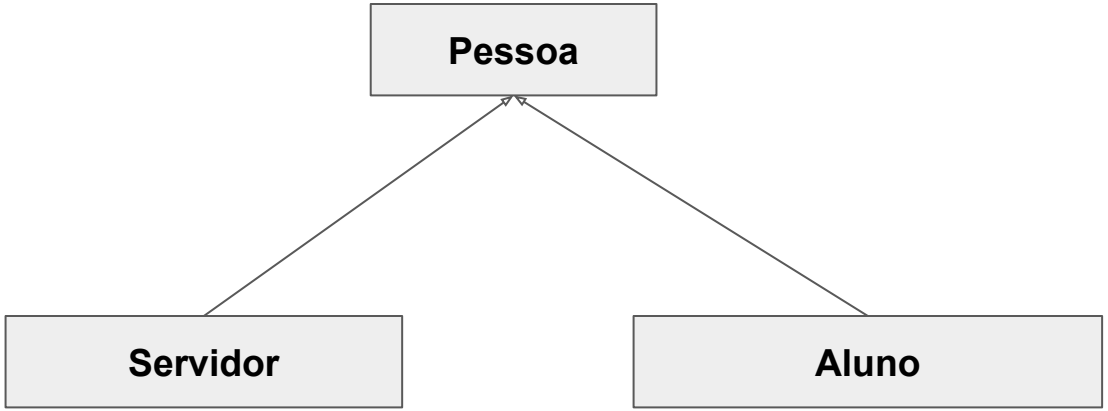


Classe básica x Classe derivada

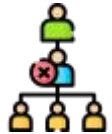
Conceitos Fundamentais



Classe básica x Classe derivada



Conceitos Fundamentais



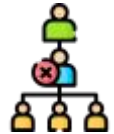
Hierarquia de Classes

A herança pode acontecer em tantos níveis quantos forem necessários

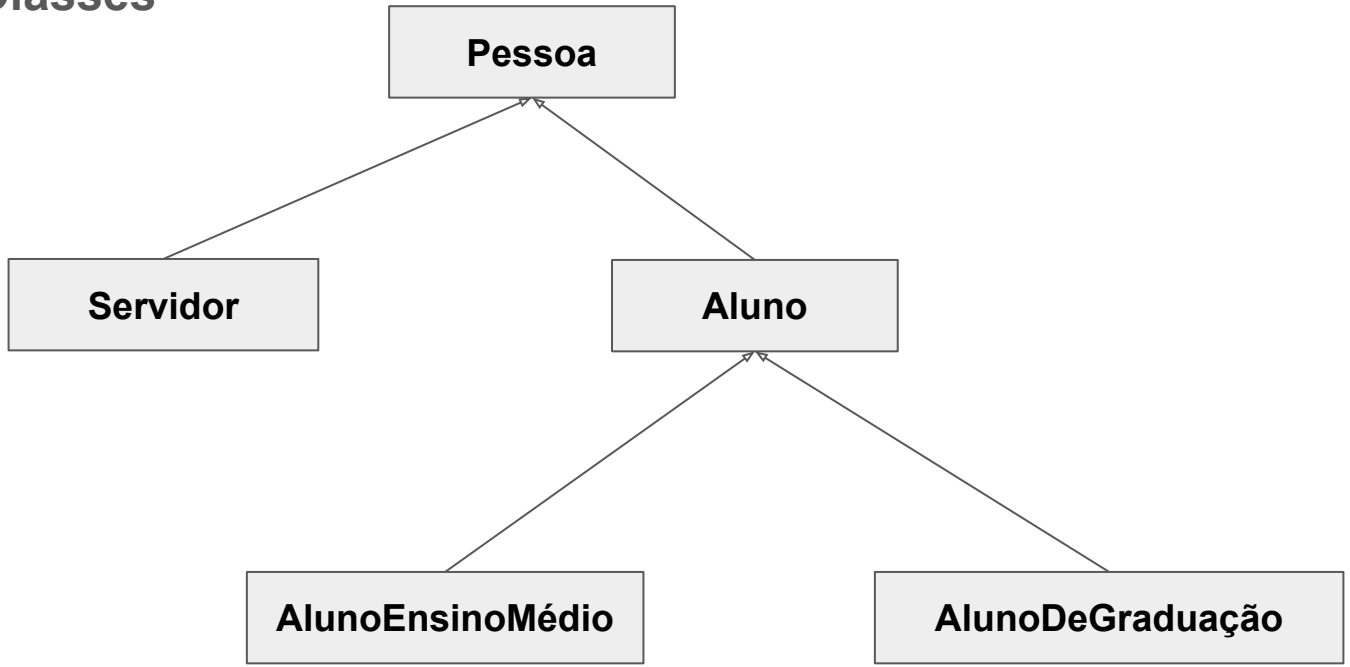
Tem-se uma **classe básica direta** quando uma classe herda diretamente essa classe básica.

Tem-se uma **classe básica indireta** quando uma classe herda uma classe que herdou a classe básica em um ou mais níveis

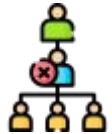
Conceitos Fundamentais



Hierarquia de Classes



Conceitos Fundamentais

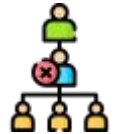


C++ permite herança múltipla

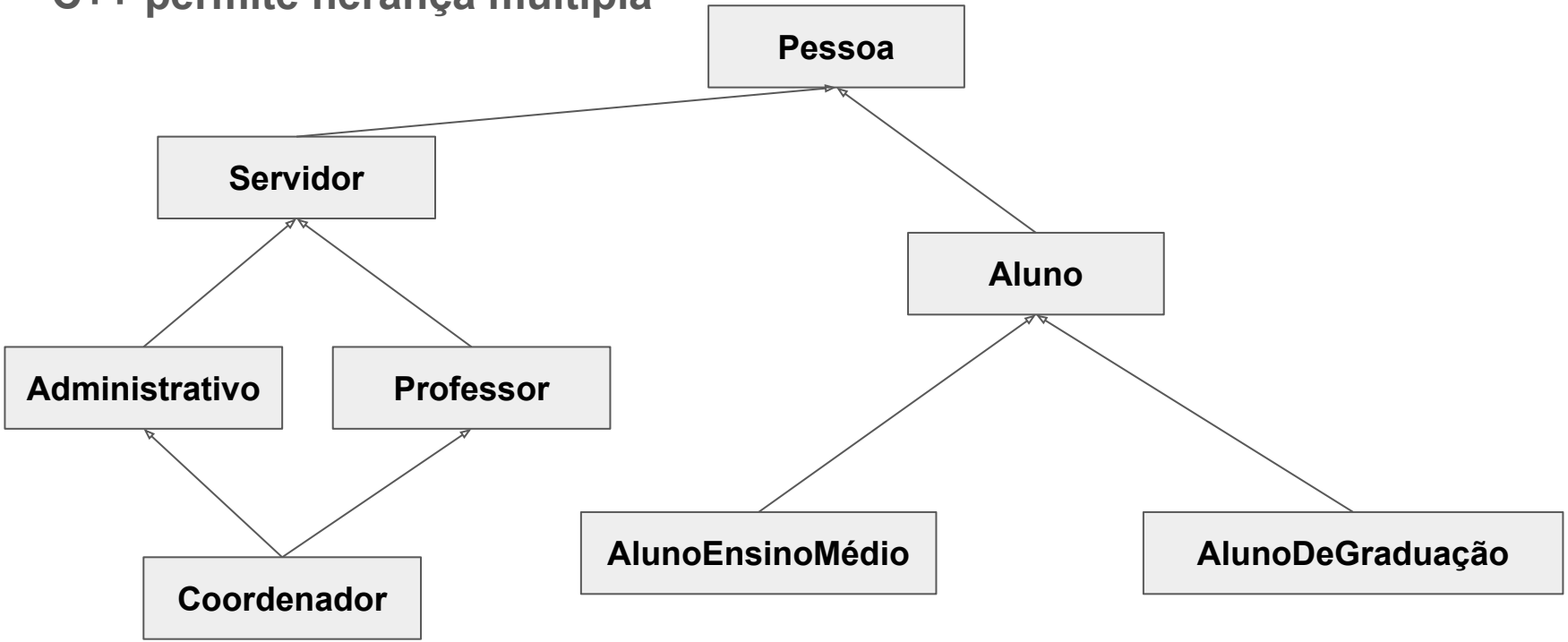
Herança simples: uma Classe herda de uma e somente uma Classe

Herança múltipla: uma Classe herda de múltiplas Classes

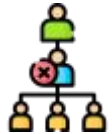
Conceitos Fundamentais



C++ permite herança múltipla



Conceitos Fundamentais



Relacionamento “é um”

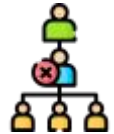
Vimos que:

a **composição** é um relacionamento do tipo **tem um**

a **agregação** é um relacionamento do tipo **contém um**

A **herança** é um relacionamento do tipo **é um**

Conceitos Fundamentais



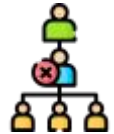
Membros protected

Já conhecemos os modificadores de acesso **public** e **private**

O modificador **protected** é usado em classes básicas e oferece um nível intermediário de acesso.

Membros protected funcionam como membros private, mas, quando a classe for herdada, os membros protected dessa classe básica podem ser acessados pela classe derivada, mas não são acessíveis de fora da classe.

Conceitos Fundamentais



Formas de herança

Em C++, a herança pode ser **public**, **protected** ou **private**.

A herança **public** é a mais comum (padrão) e é a única que será explorada

Ver tabela no final do material…

Estudo de caso

Monster

Um Monstro!!!

Estudo de caso

Monster

Monster

Um Monstro!!!

Sala:

Estendida:

Estudo de caso

Monster

GrowlingMonster

Growling Monster

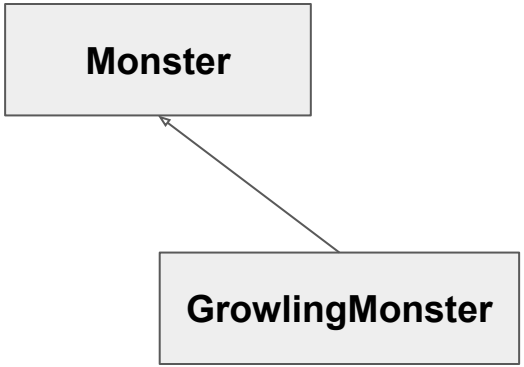
Um Monstro Grunhidor!!!

Sala: OnlineGDB
online compiler and debugger for c/c++

Estudo de caso

Growling Monster

Um Monstro Grunhidor!!!

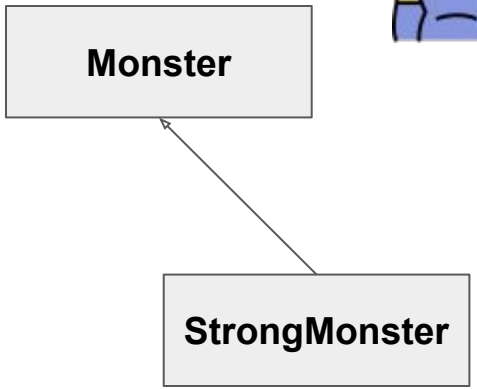


Agora com **reuso de código**… dá-lhe herança!

Sala:

OnlineGDB
online compiler and debugger for c/c++

Estudo de caso



Monster e StrongMonster

Sobrescrevendo métodos…

Sala: OnlineGDB — online compiler and debugger for c/c++

Versão estendida: OnlineGDB — online compiler and debugger for c/c++

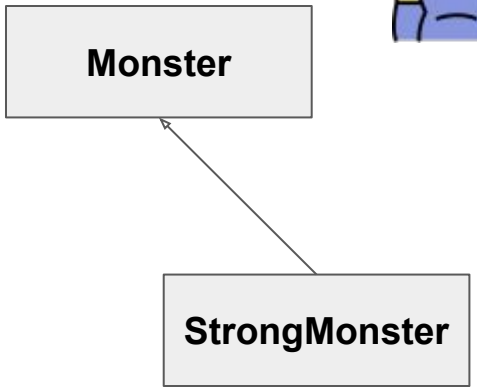
Estudo de caso

Monster e StrongMonster

Quando uma classe derivada possui um método com a mesma assinatura de um método da classe base, diz-se que há uma ***sobrescrita de métodos****.*

A **sobrescrita** faz com que o novo comportamento (método) implementado na derivada substitua o comportamento existente na classe básica.

Estudo de caso



Monster e StrongMonster

Usando **membros protected**

Sala: OnlineGDB
online compiler and debugger for c/c++

Versão estendida: OnlineGDB
online compiler and debugger for c/c++

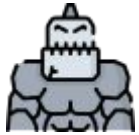
Estudo de caso

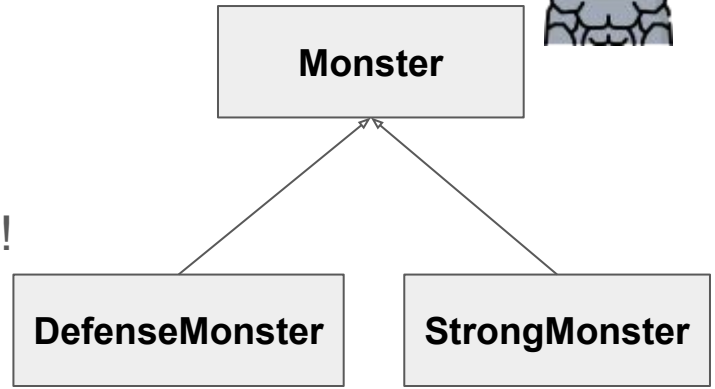
Monster e StrongMonster

Membros protected

Classes derivadas podem acessar os membros protected da classe base. Nem sempre isso é ideal. Depende do design. Há quem diga que quebra o encapsulamento.

Estudo de caso





Monster, StrongMonster e DefenseMonster

Que tal adicionar um Monstro mais defensivo!?!

Sala: OnlineGDB
online compiler and debugger for c/c++

Estendida: OnlineGDB
online compiler and debugger for c/c++

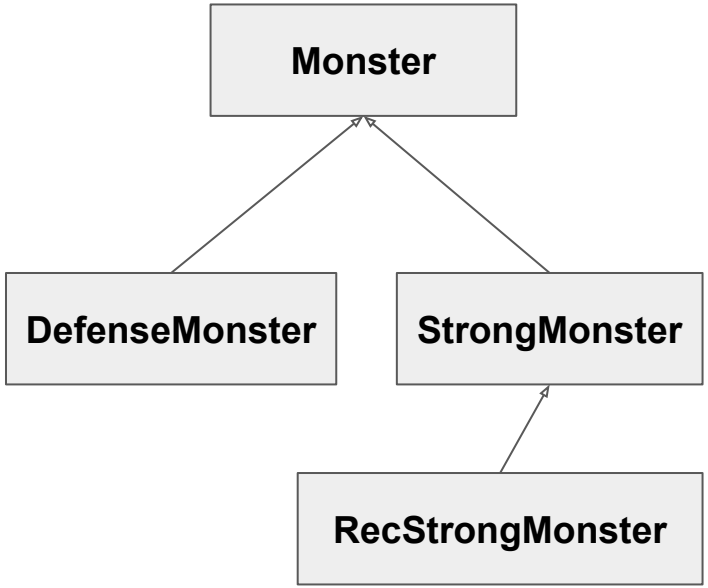
Estudo de caso

Monster, StrongMonster, DefenseMonster e RecStrongMonster

Descendo mais um nível na hierarquia. Um StrongMonster que recupera seu life quando golpeia!!!

Sala: OnlineGDB
online compiler and debugger for c/c++

Estendida: OnlineGDB
online compiler and debugger for c/c++



Análise técnica de construtores e destrutores

Monster, StrongMonster, DefenseMonster e RecStrongMonster

Analisando ordem de chamadas aos construtores e destrutores.

Um conceito muito, mas muito importante…

IMPORTANTE!!!!

Isto é fundamental:

Uma referência (ou ponteiro) de uma classe base também pode ser usada para referenciar (ou apontar) objetos de classes derivadas. Contudo a referência (ou ponteiro) só poderá acessar "o que ela conhece", isto é, membros da classe base que ela referencia (ou aponta).

Herança public, protected e private

Especificador de acesso de membro de classe básica	Tipo de herança		
	Herança `public`	Herança `protected`	Herança `private`
`public`	`public` na classe derivada. Pode ser acessada diretamente por funções-membro, funções `friend` e funções não-membro.	`protected` na classe derivada. Pode ser acessada diretamente por funções-membro e funções `friend`.	`private` na classe derivada. Pode ser acessada diretamente por funções-membro e funções `friend`.
`protected`	`protected` na classe derivada. Pode ser acessada diretamente por funções-membro e funções `friend`.	`protected` na classe derivada. Pode ser acessada diretamente por funções-membro e funções `friend`.	`private` na classe derivada. Pode ser acessada diretamente por funções-membro e funções `friend`.
`private`	Oculta na classe derivada. Pode ser acessada por funções-membro e funções `friend` por meio das funções-membro `public` ou `protected` da classe básica.	Oculta na classe derivada. Pode ser acessada por funções-membro e funções `friend` por meio das funções-membro `public` ou `protected` da classe básica.	Oculta na classe derivada. Pode ser acessada por funções-membro e funções `friend` por meio das funções-membro `public` ou `protected` da classe básica.

Herança public, protected e private

Resumindo…

As heranças protected e private não são relacionamentos “é um”!!!!!

Considere usar sempre herança public, a menos que saiba o que está fazendo.


LEVEL
UP