Apostila de Programação Orientada a Objetos

November 28, 2018

Contents

01 - Introdução

Motivação

Linguagem C

Linguagem C++

Programação Orientada à Objetos

02 - Structs

Struct em C

Pilhar de OO - Abstração

Struct em C++

Atributos

Métodos

Construtores

Destrutores

Operadores

Exercícios

03 - Class em C++

Pilhar de OO - Ecapsulamento

Modificadores de acesso

Public

Private

Protected

Exercícios

Pilhar de OO - Herança

Herança 01

Herança é utilizada para fazer com que uma classe herde características de outra(s). Utilizando herança, podemos utilizar características de uma classe em outra sem ter que codificar novamente as características.

Uma analogia que podemos fazer é entre um animal e um humano. Como todo humano também é um animal, podemos reaproveitar todas as características de um animal na criação de um humano. Quando uma classe A herda característica de uma classe B, dizemos que a classe A é uma classe derivada de B e que a classe B é a classe base de A.

Para utilizar herança em C++, devemos especificar qual a visibilidade da herança e qual a classe base. O modificar de visibilidade vai alterar a visibilidade dos atributos da classe base na classe derivada (Características ocultas não podem ser acessadas diretamente, é necessário utilizar métodos de classes herdadas para conseguir acessar a características).

• public: As características públicas e protegidas são herdadas sem mudança de visibilidade. Características privadas serão ocultas. • protected: As características públicas e protegidas são herdadas protegidas e as características privadas serão ocultas. • private (default): As características públicas são herdadas privadas e as características protegidas e privadas serão ocultas.

Em C++, podemos definir A sintaxe para utilizar herança em C++ é: class <$classe_derivada$ >:< $visibilidade$ >< $classe_base$ >

Ex: A classe B herda da classe A de forma pública, isso faz com que a classe B também tenha os atributos inteiros a e b com visibilidade protected. class A protected: int a, b; class B: public A

Exercícios: 1) Implemente duas classes, animal e humano: a) Animal: Apresenta os atributos protegidos: idade e sexo. b) Humano: Herda de animal de forma pública. Apresenta os atributos protegidos: nome e altura. 4 Getters e 4 Setters para os 4 atributos do humano (Quando humano herda de animal, o humano recebe os atributos idade e sexo como heraça). c) Crie um objeto Humano e utilize seus métodos para verificar o bom funcionamento da sua implementação. Link resposta: COLOCAR AQUI

2) Implemente dois classes: veículo e carro. a) Veículo: apresenta os atributos protegidos: quantidade atual de passageiros, velocidade atual. b) Carro que herda de forma pública: Apresenta construtor público sem parâmetro que inicializa a quantidade atual de passageiros com 1 e velocidade atual com 0. Método público para aumentar a velocidade. Métodos públicos para aumentar ou diminuir a velocidade atual. Métodos públicos para aumentar ou diminuir um passageiro. Método público para imprimir todos os atributos do carro. Não permita que aconteça alteração no número de passageiros se a velocidade atual do carro não for 0, pois um passageiro não vai pular de um carro em movimento. Não permita que o número de passageiros ou velocidade assuma valores negativos. Não permita que um carro tenha velocidade positiva se não possuir, pelo menos, um passageiro, pois o carro precisa de um motorista para acelera-lo. c) Crie um objeto Humano e utilize seus métodos para verificar o bom funcionamento da sua implementação. Link resposta: COLOCAR AQUI

Escopo

Vimos que ao utilizar herança, uma classe derivada recebe características de uma classe base, porém, o que acontece caso a classe derivada possua características de mesmo nome da classe base?

Quando uma classe possui mais de uma característica de mesmo nome, ela vai guardar todas, porém o acesso será diferenciado para cada uma das características. Observe o exemplo a seguir. Uma classe B herda um inteiro x da classe A e a classe C herda da classe B. Como a classe B possui dois inteiros x (um do escopo de A e um do escopo de B), a classe C vai possuir 3 inteiros x.

class A public: int x; ; class B: public A public: int x; ; class C:public B public: int x; ;

Se a classe C possui três atributos inteiro x, como podemos especificar o acesso de cada um? Na linguagem C/C++, sempre que existe mais de uma variável com um mesmo nome que podem ser acessadas em um dado escopo, a linguagem acessa a variável local. Ou seja, se um objeto do tipo C tentar acessar um atributo x diretamente,

ele vai acessar o x do escopo do C, e de forma semelhante, se um objeto do tipo B tentar acessar um atributo x diretamente, ele vai acessar o x do escopo do B. Para um objeto do tipo C conseguir acessar os atributos x do escopo de A e B, temos duas opções: 1. Utilizar métodos das classes A e B: Como a prioridade será para o atributo local, se as classes A e B possuirem métodos para acessar os atributos, ao utilizar esses métodos vamos conseguir acessar os atributos. Link do Código: COLOCAR AQUI. 2. Utilizar operadores de escopo: Uma forma mais simples, é especificar qual delas vamos acessar utilizando o operador de escopo (::). Como a classe C possui uma classe B e uma classe B possui uma classe A, "dentro" da classe C existe uma classe B e uma A. Para acessar diretamente o que está no escopo de A, utilizamos A::, e para acessar diretamente o que está no escopo da B, utilizamos B::. Link do Código: COLOCAR AQUI.

Visibilidade da Herança

Exemplos de Herança

Herança Múltipla

Exercícios

Pilhar de OO - Polimorfismo

Herança 02

Sobrescrita de métodos

Resolução de exercícios utilizando herança

Agenda telefônica

Nesse exercícios vamos implementar uma lista telefônica que vai armazenar uma lista de pessoas. Para implementar essa lista, vamos utilizar as seguinte classes: Pessoa, Nó, Lista e Agenda.

A classe Pessoa deve possuir:

- Atributos protegidos:
  - Nome (string)
  - Telefone (string)
- Métodos públicos:
  - Get/Set para os atributos
  - Construtor sem parâmetro que inicializa os atributos com strings vazias.
  - Construtor que recebe duas strings como parâmetro e inicializa os dois atributos.

A classe No deve possuir:

- Atributos públicos:
  - Conteúdo (Pessoa)
  - Próximo (Ponteiro para Nó)
  - Anterior (Ponteiro para Nó)
- Métodos públicos:
  - Construtor sem parâmetro que inicializa os ponteiros como nulos.
  - Construtor que recebe uma Pessoa como parâmetro, inicializa o Conteúdo com a Pessoa recebida e inicializa os ponteiros como nulos.

A classe Lista deve possuir:

- Atributos protegidos:
  - Inicial (Ponteiro para Nó)
  - Final (Ponteiro para Nó)
  - Tamanho (Inteiro)
- Métodos públicos:
  - Lista(): Inicializa os ponteiros como nulos e o tamanho como 0.
  - Adicionar no final(Pessoa x): Adiciona a pessoa X ao final da Lista.
  - Adiciona no inicio(Pessoa x): Adiciona a pessoa X ao inicio da Lista.

A classe Agenda deve possuir:

- Atributos protegidos:
  - Pessoas (Lista)
- Métodos públicos:
  - Ordena(): Ordena a Lista por ordem lexicográfica.
  - Imprime(): Imprime a Lista.

Solução disponível no **link**.