Federal University of Santa Catarina
Real-Time Operating Systems

Introduction to Object-Oriented Programming with C++

Prof. Dr. Giovani Gracioli
giovani@lisha.ufsc.br
http://www.lisha.ufsc.br/Giovani

Objectives

- Introduce inheritance in C++
- How to create a class that inherits from another one
- Base class and derived class concepts

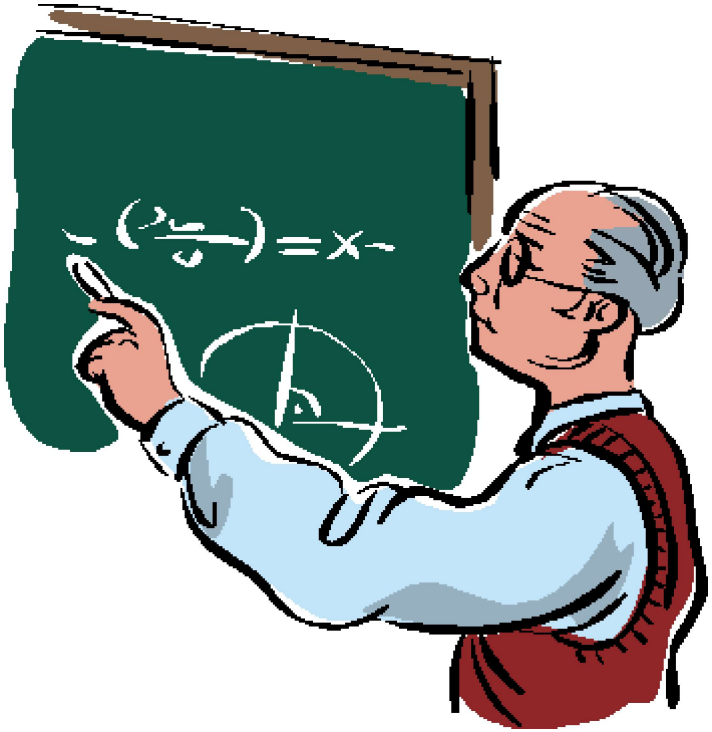
What you need to know to follow

- C++ skills
  - Concepts of class and object
  - How to declare a class in C++
  - How to create an object in C++
  - How to use an object in C++
  - How to use constructors and destructors

What you will learn

- Understand what inheritance is in an OO language
- Declare base classes and derived classes

Let's get started



Introduction

- Inheritance is a form of SW reuse in which you create a class that absorbs an existing class's capabilities, then customize or enhances them
- You can specify that a new class should inherit the members (attribute and methods) of an existing class
- The existing class is called base class and the new class is called derived class

Introduction

- Derived classes represents more specialized groups of objects
- A derived class contains inherited behaviors from its base class and also additional behaviors
- A derived class may also personalize inherited behaviors from the base classes

Base class and derived class examples

- Every derived-class object is an object of its base class
- One base class can have many derived classes => the set represented by the base class is larger than of its derived classes
- Inheritance relationships form class hierarchies

Base class	Derived classes
Student	GraduateStudent, UndergraduateStudent
Shape	Circle, Triangle, Rectangle, Sphere, Cube
Loan	CarLoan, HomeImprovementLoan, MortgageLoan
Employee	Faculty, Staff
Account	CheckingAccount, SavingsAccount

Single and multiple inheritance

- Single inheritance
  - A class is derived from *one* base class
- Multiple inheritance
  - A derived class inherits from *two or more* base classes (possibly unrelated)

Example of class hierarchy

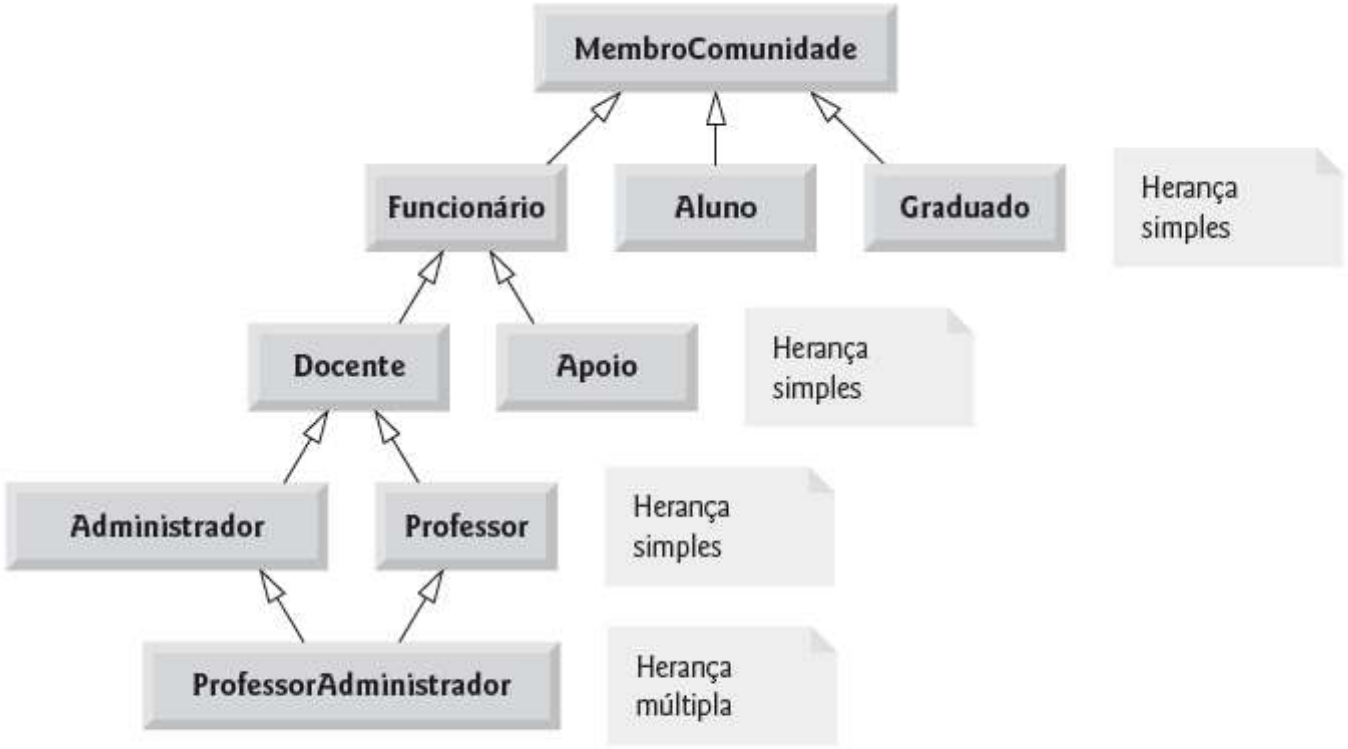


Figura 20.2 ■ Hierarquia de herança de uma universidade `MembroComunidade`.

protected, private, and public members

- A base class's public members are accessible within its body and anywhere that the program has a handle (name, reference, or pointer)
- A base class's private members are accessible only within its body and to the friends of that base class
- protected access offers and intermediate level of protection
  - Members can be accessed within the body, by members of friends of that base class and by members and friends of any derived classes

protected, private, and public members

- When a derived-class member function redefines a base-class member function, the base-class member function can still be accessed from the derived class by using the scope resolution operator (::)

Summary of accessibility

Base-class member-access specifier	Type of inheritance		
	`public` inheritance	`protected` inheritance	`private` inheritance
`public`	`public` in derived class. Can be accessed directly by member functions, `friend` functions and nonmember functions.	`protected` in derived class. Can be accessed directly by member functions and `friend` functions.	`private` in derived class. Can be accessed directly by member functions and `friend` functions.
`protected`	`protected` in derived class. Can be accessed directly by member functions and `friend` functions.	`protected` in derived class. Can be accessed directly by member functions and `friend` functions.	`private` in derived class. Can be accessed directly by member functions and `friend` functions.
`private`	Hidden in derived class. Can be accessed by member functions and `friend` functions through `public` or `protected` member functions of the base class.	Hidden in derived class. Can be accessed by member functions and `friend` functions through `public` or `protected` member functions of the base class.	Hidden in derived class. Can be accessed by member functions and `friend` functions through `public` or `protected` member functions of the base class.

Fig. 12.16 | Summary of base-class member accessibility in a derived class.

Declaring a derived class

- Example:

```
class derived-class-name : public base-class-name {
    //body of the derived class
};
```

- Note the use of the operator ":", which promotes the inheritance between the two classes
- Before the base-class name, we must declare the visibility of the base class (public, private, or protected)

Example: road vehicles

```
#include <iostream>
using namespace std;
class VeiculoRodoviario // Define uma classe base veículos.
{
 private:
        int rodas;
        int passageiros;
 public:
       VeiculoRodoviario() { }
        VeiculoRodoviario(int r, int p) {
              rodas = r;
              passageiros = p;
        }
        void setRodas(int num) { rodas = num; }
        int getRodas() { return rodas; }
        void setPassageiros(int num) { passageiros = num; }
        int getPassageiros() { return passageiros; }
};
class Caminhao : public VeiculoRodoviario // Define um caminhao.
{
        int carga;
 public:
        Caminhao() { }
        Caminhao(int c, int r, int p) : VeiculoRodoviario(r, p) { carga = c; }
        void setCarga(int size) { carga = size; }
        int getCarga() { return carga; }
        void mostrar() {
              cout << "rodas: " << getRodas() << "\n";
              cout << "passageiros: " << getPassageiros() << "\n";
              cout << "carga (capacidade em litros): " << getCarga() << "\n";
        }
};
```

Example: road vehicles

```
enum tipo {car, van, vagao};

class Automovel : public VeiculoRodoviario // Define um automovel.
{
        enum tipo tipoCarro;
 public:
       Automovel() { }
       Automovel(tipo t, int r, int p) : VeiculoRodoviario(r, p) { tipoCarro = t; }
        void setTipo(tipo t) { tipoCarro = t; }
        enum tipo getTipo() { return tipoCarro; }

        void mostrar() {
         cout << "rodas: " << getRodas() << "\n";
         cout << "passageiros: " << getPassageiros() << "\n";
         cout << "tipo: ";
         switch(getTipo())
         {
                case van: cout << "van\n";
                          break;
                case car: cout << "carro\n";
                          break;
                case vagao: cout << "vagao\n";

         }
       }
};
```

Example: road vehicles

```
int main()
{
    Caminhao t1, t2(6, 4, 2000);
    Automovel c;
    t1.setRodas(18);
    t1.setPassageiros(2);
    t1.setCarga(3200);
    t1.mostrar();
    cout << "\n";
    t2.mostrar();
    cout << "\n";
    c.setRodas(4);
    c.setPassageiros(6);
    c.setTipo(van);
    c.mostrar();
    return 0;
}
```

Constructors and Destructors in Derived Classes

- Instantiating a derived-class object begins a chain of constructor class
  - The derived-class constructor, before performing its own tasks, invokes its direct base class's constructor either explicitly or implicitly
- The last constructor called in the chain is the one of the class at the base of the hierarchy, whose body actually finishes executing first
- The destructors execute in reserve order

Tasks

- Read chapter 12 of the text book (C++ how to program 8th edition)

- Exercises of chapter 12
  - 12.3, 12.4, 12.6, 12.7, 12.9

References

- Paul Deitel e Harvey Deitel, C++: como programar, 5a edição, Ed. Prentice Hall Brasil, 2006.
- Paul Deitel e Harvey Deitel, C++: how to program, 8th edition, Ed. Prentice Hall, 2012.