Curso de Modelagem e Teste de Software Embarcado Automotivo

Introdução a Orientação a Objetos

Prof. Dr. Giovani Gracioli
giovani@lisha.ufsc.br

Prof. Dr. Antônio Augusto Fröhlich
guto@lisha.ufsc.br

ROTA2030
FUNDEP

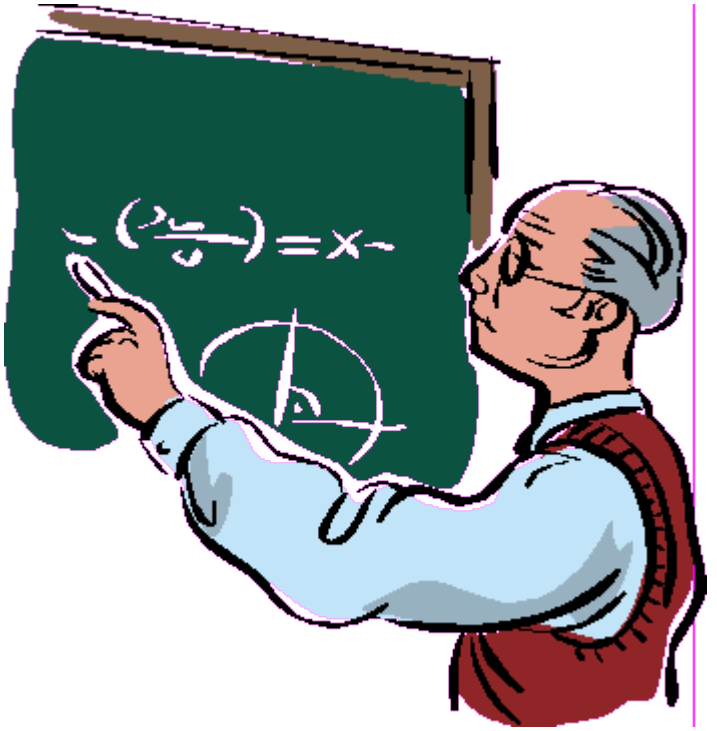
Objectives

- Introduction to OO with C++
  - Improved C, with more functionalities and with object orientation
- Present concepts of classes and objects
- Explain how to create classes and objects with C++
- Demonstrate how to use classes and objects in C++

What you will learn

- Use I/O in C++
- To define a class and use it to create an object
- To define methods and declare attributes of a class
- To call methods to do a task
- To use class constructors to initialize data of an object when it is created

Let's get started



C++ (1)

- C++ improves many resources from the C language and offers the paradigm of Object-Oriented Programming
  - Increases productivity, quality, and re-usability of software
- C++ was developed by Björn Stroustrup, in the Bell Laboratories, and originally was called of “C with classes”
- The name C++ includes the increment operator of C (++) to indicate that C++ is an improved version of C

C++ (2)

- The file names in C have the extension .c (lower case)
- The file names in C++ can have several extensions, like .cpp, .cxx, or .cc
- C++ supports many commands from C
  - while, for, if, switch, do..while, etc..
- And also types
  - int, float, double, struct, enum, pointers, arrays, etc
  - However, the use of libraries is different, example...

C++: I/O example

```
1  // Figura 15.1: fig15_01.cpp
2  // Programa de adição que mostra a soma de dois números.
3  #include <iostream> // permite que o programa realize entrada e saída
4
5  int main()
6  {
7     int number1; // primeiro inteiro a somar
8
9     std::cout << "Digite o primeiro inteiro: "; // pede dados do usuário
10    std::cin >> number1; // lê primeiro inteiro do usuário em number1
11
12    int number2; // segundo inteiro a somar
13    int sum; // soma de number1 e number2
14
15    std::cout << "Digite o segundo inteiro: "; // pede dados do usuário
16    std::cin >> number2; // lê segundo inteiro do usuário em number2
17    sum = number1 + number2; // soma os números; armazena resultado em sum
18    std::cout << "A soma é " << sum << std::endl; // mostra a soma; fim da linha
19 } // fim da função main
```

```
Digite primeiro inteiro: 45
Digite segundo inteiro: 72
A soma é 117
```

Figura 15.1 ■ Programa de adição que exibe a soma de dois números.

Input and Output (1)

- Line 9 uses the standard output stream object – std::cout – and the stream insertion operator, <<, to exhibit the string “Digite o primeiro inteiro: “
- The I/O in C++ are performed by streams of characters
- Thus, when line 9 is executed, it sends the stream “Digite o primeiro inteiro: ” to std::cout, which normally is “connected” to the screen

Input and Output (2)

- Line 10 uses the standard input stream object – std::cin – and the operator of stream extraction, >>, to obtain a value from the keyboard

- Cascade output

```
std::cout << “A soma é " << number1 + number2 << std::endl;
```

- Or

```
std::cout << “A soma é " << number1 + number2 << “\n”;
```

Standard Library of C++

Arquivo de cabeçalho da biblioteca-padrão de C++	Explicação
<cmath>	Contém protótipos para as funções da biblioteca matemática. Esse arquivo de cabeçalho substitui o arquivo de cabeçalho <math.h>.
<cstdlib>	Contém protótipos de função para conversões de números para texto, texto para números, alocação de memória, números aleatórios e outras funções utilitárias diversas. Esse arquivo de cabeçalho substitui o arquivo de cabeçalho <stdlib>.
<ctime>	Contém protótipos de função e tipos para a manipulação de hora e data. Esse arquivo de cabeçalho substitui o arquivo de cabeçalho <time.h>.
<vector>, <list>, <deque>, <queue>, <stack>, <map>, <set>, <bitset>	Esses arquivos de cabeçalho contêm classes que implementam os contêineres da biblioteca-padrão de C++. Os contêineres armazenam dados durante a execução de um programa.
<cctype>	Contém protótipos para as funções que testam caracteres em busca de certas propriedades (por exemplo, se o caractere é um dígito ou um sinal de pontuação) e protótipos para as funções que podem ser usadas para converter letras minúsculas em maiúsculas e vice-versa. Esse arquivo de cabeçalho substitui o arquivo de cabeçalho <ctype.h>.
<cstring>	Contém protótipos para funções de processamento de strings no estilo de C. Esse arquivo de cabeçalho substitui o arquivo de cabeçalho <string.h>.
<typeinfo>	Contém classes para identificação de tipo em tempo de execução (determinando tipos de dados no tempo de execução).
<exception>, <stdexcept>	Esses arquivos de cabeçalho contêm classes que são usadas para tratamento de exceção (discutido no Capítulo 24).
<memory>	Contém classes e funções usadas pela biblioteca-padrão de C++ para alocar memória aos contêineres da biblioteca-padrão de C++. Esse cabeçalho é usado no Capítulo 24.
<fstream>	Contém protótipos para funções que realizam entrada de arquivos no disco e saída para arquivos no disco. Esse arquivo de cabeçalho substitui o arquivo de cabeçalho <fstream.h>.
<string>	Contém a definição da classe string da biblioteca-padrão de C++.
<sstream>	Contém protótipos para as funções que realizam entrada de strings na memória e saída para strings na memória.

Function Overloading (1)

- C++ allows the definition of functions with the same name (they must have different parameters, like types, number, or order)
- This capacity is called function overloading. When a overloaded function is called, the C++ compiler selects the appropriate function by analyzing the number, types, and order of the parameters
- The function overload is used to create functions that do similar tasks with different types

Function Overloading (2)

```
1  // Figura 15.10: fig15_10.cpp
2  // Funções sobrecarregadas.
3  #include <iostream>
4  using namespace std;
5
6  // função square para valores int
7  int square( int x )
8  {
9     cout << "quadrado do int " << x << " é ";
10    return x * x;
11 } // fim da função square com argumento int
12
13 // função square para valores double
14 double square( double y )
15 {
```

Figura 15.10 ■ Funções square sobrecarregadas. (Parte 1 de 2.)

Function Overloading (3)

```
16    cout << "quadrado do double " << y << " é ";
17    return y * y;
18 } // fim da função square com argumento double
19
20 int main()
21 {
22    cout << square( 7 ); // chama versão int
23    cout << endl;
24    cout << square( 7.5 ); // chama versão double
25    cout << endl;
26 } // fim do main
```

```
quadrado do int 7 é 49
quadrado do double 7.5 é 56.25
```

Figura 15.10 ■ Funções square sobrecarregadas. (Parte 2 de 2.)

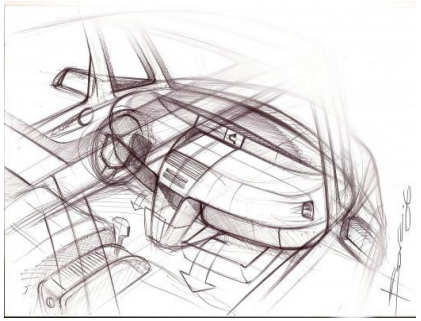
Introduction to Classes and Objects

- Classes and objects are two fundamental concepts of any object-oriented programming language

- Normally, your C++ programs will consist of a main function and one or more classes, each one containing member functions (methods) and data members (attributes)

Classes, objects, methods, and attributes (1)

- Analogy: suppose you want to drive a car faster stepping on the throttle
  - First of all, the car must be designed and built
  - The pedal “hides” the complex mechanisms that make the car move fast, the wheel “hides” the mechanisms that make the car turn
  - This allows people that do not know the internals of a car to drive it easily, just using pedals, breaking, wheel, and other simple “interfaces”



Classes, objects, methods, and attributes (2)

- In C++, classes are used to house a function, as “drawings” in engineering are used to house the project of the throttle pedal

- The function of a class is called member function or method
  - They do the tasks of classes

- As you cannot drive a draw, you cannot “drive” a class either
  - It is necessary to create an object
  - Many objects can be created from the same class, as many cars are built from the same draw

Classes, objects, methods, and attributes (3)

- When a person steps on the throttle, a message is sent to the system that controls the car speed
- Similarly, messages are sent to objects (methods call)
  - Request of a service of an object

Classes, objects, methods, and attributes (4)

- In addition to the functionalities that a car offers, it also has several attributes, like color, number of doors, KM, etc
  - These attributes are also represented in the car's project
  - Each car has its own attributes

- Similarly, each object has attributes that are loaded when it is created
  - These attributes or data members are specified as part of the object

Example: how to create a class

```
1  // Fig. 16.1: fig16_01.cpp
2  // Define a classe GradeBook com uma função-membro displayMessage,
3  // cria um objeto GradeBook e chama sua função displayMessage.
4  #include <iostream>
5  using namespace std;
6
7  // Definição da classe GradeBook
8  class GradeBook
9  {
10 public:
11    // função que mostra uma mensagem de boas-vindas ao usuário de GradeBook
12    void displayMessage()
13    {
14       cout << "Bem-vindo ao Grade Book!" << endl;
15    } // fim da função displayMessage
16 }; // fim da classe GradeBook
17
18 // função main inicia a execução do programa
19 int main()
20 {
21    GradeBook myGradeBook; // cria um objeto GradeBook chamado myGradeBook
22    myGradeBook.displayMessage(); // chama função displayMessage do objeto
23 } // fim do main
```

```
Bem-vindo ao Grade Book!
```

Figura 16.1 ■ Define a classe GradeBook com uma função-membro displayMessage, cria um objeto GradeBook e chama sua função displayMessage.

Defining a class with a Method

- Normally, you cannot call a class method until creating an object of this class (line 21)
  - The variable type is GradeBook
  - It is a user-defined type
  - Each created class is a new type that can be used to create objects

- In line 22, the method displayMessage is called using the object myGradeBook, followed by point operator (.)

Defining a method with a parameter (1)

```
1  // Fig. 16.3: fig16_03.cpp
2  // Define classe GradeBook com função-membro que usa um parâmetro;
3  // Cria um objeto GradeBook e chama sua função displayMessage.
4  #include <iostream>
5  #include <string> // programa usa classe de string padrão de C++
6  using namespace std;
7
8  // Definição da classe GradeBook
9  class GradeBook
10 {
11 public:
12    // função que mostra mensagem de boas-vindas ao usuário do GradeBook
13    void displayMessage( string courseName )
14    {
15       cout << "Bem-vindo ao grade book para\n" << courseName << "!"
16          << endl;
17    } // fim da função displayMessage
18 }; // fim da classe GradeBook
19
20 // função main inicia a execução do programa
21 int main()
22 {
```

Defining a method with a parameter (2)

```
23      string nameOfCourse; // string de caracteres para armazenar o nome do curso
24      GradeBook myGradeBook; // cria um objeto GradeBook chamado myGradeBook
25
26      // pede e insere nome do curso
27      cout << "Favor informar o nome do curso:" << endl;
28      getline( cin, nameOfCourse ); // lê um nome de curso com espaços
29      cout << endl; // mostra uma linha em branco
30
31      // chama função displayMessage de myGradeBook
32      // e passa nameOfCourse como argumento
33      myGradeBook.displayMessage( nameOfCourse );
34  } // fim do main
```

```
Favor informar o nome do curso:
CS101 Introdução à programação C++

Bem-vindo ao grade book para
CS101 Introdução à programação C++!
```

Figura 16.3 ■ Define classe GradeBook com uma função-membro que usa um parâmetro, cria um objeto GradeBook e chama sua função displayMessage.

Attributes, set and get methods (1)

- Variables declared inside a function are known as local variables
  - Must be declared before their usage, are not accessed outside the function, and are “destroyed” when the function ends
- Attributes of an object are represented as variables in a class definition
  - They are declared inside the class definition, but outside the methods
  - Methods manipulate the attributes of an object
- Each object has its own copy of the attributes in memory

Attributes, set and get methods (2)

```
1  // Fig. 16.5: fig16_05.cpp
2  // Define classe GradeBook que contém um dado-membro courseName
3  // e funções-membro para definir e obter seu valor;
4  // Cria e manipula um objeto GradeBook com essas funções.
5  #include <iostream>
6  #include <string> // programa usa classe string-padrão da C++
7  using namespace std;
8
9  // Definição da classe GradeBook
10 class GradeBook
11 {
12 public:
13    // função que define o nome do curso
14    void setCourseName( string name )
15    {
16       courseName = name; // armazena o nome do curso no objeto
17    } // fim da função setCourseName
18
19    // função que obtém o nome do curso
20    string getCourseName()
21    {
22       return courseName; // retorna o courseName do objeto
23    } // fim da função getCourseName
24
25    // função que mostra uma mensagem de boas-vindas
26    void displayMessage()
27    {
28       // essa instrução chama getCourseName para obter o
29       // nome do curso que esse GradeBook representa
30       cout << "Bem-vindo ao grade book para\n" << getCourseName() << "!"
```

Figura 16.5 ■ Definições e teste da classe GradeBook com um dado-membro e funções *set* e *get*. (Parte 1 de 2.)

Attributes, set and get methods (3)

```
31          << endl;
32       } // fim da função displayMessage
33    private:
34       string courseName; // nome do curso para esse GradeBook
35    }; // fim da classe GradeBook
36
37    // função main inicia a execução do programa
38    int main()
39    {
40       string nameOfCourse; // string de caracteres para armazenar o nome do curso
41       GradeBook myGradeBook; // cria um objeto GradeBook chamado myGradeBook
42
43       // exibe valor inicial de courseName
44       cout << "Nome inicial do curso é: " << myGradeBook.getCourseName()
45          << endl;
46
47       // solicita, insere e define nome do curso
48       cout << "\nFavor digitar o nome do curso:" << endl;
49       getline( cin, nameOfCourse ); // lê um nome de curso com espaços
50       myGradeBook.setCourseName( nameOfCourse ); // define o nome do curso
51
52       cout << endl; // gera uma linha em branco
53       myGradeBook.displayMessage(); // exibe mensagem com novo nome do curso
54    } // fim do main
```

```
Nome inicial do curso é:

Favor digitar o nome do curso:
CS101 Introdução à programação C++

Bem-vindo ao grade book para
CS101 Introdução à programação C++!
```

Figura 16.5 ■ Definições e teste da classe GradeBook com um dado-membro e funções *set* e *get*. (Parte 2 de 2.)

Initializing objects with constructors (1)

- Each class must offer a constructor that can be used to initialize an object of that class when it is created
- A constructor is a special method that must be defined with the same name of the class (thus the compiler knows it is a constructor)
- Constructors cannot return values

Initializing objects with constructors (2)

- C++ requires a call to a constructor for each created object, which helps in keeping the object initialization before its usage
- The call occurs implicitly when the object is created
- If a class does not explicitly include a constructor, the compiler will include a standard constructor – a constructor with no parameters

Initializing objects with constructors (3)

```
4  // quando cada objeto GradeBook for criado.
5  #include <iostream>
6  #include <string> // programa usa classe de string C++ padrão
7  using namespace std;
8
9  // Definição de classe GradeBook
10 class GradeBook
11 {
12 public:
13    // construtor inicializa courseName com string fornecida como argumento
14    GradeBook( string name )
15    {
16       setCourseName( name ); // chama função set para inicializar courseName
17    } // fim do construtor GradeBook
18
19    // função para definir o nome do curso
20    void setCourseName( string name )
21    {
22       courseName = name; // armazena o nome do curso no objeto
23    } // fim da função setCourseName
24
25    // função para obter o nome do curso
26    string getCourseName()
27    {
28       return courseName; // retorna courseName do objeto
29    } // fim da função getCourseName
30
31    // exibe mensagem de boas-vindas para o usuário do GradeBook
32    void displayMessage()
```

Initializing objects with constructors (4)

```
33    {
34       // chama getCourseName para obter o courseName
35       cout << "Bem-vindo ao grade book para\n" << getCourseName()
36          << "!" << endl;
37    } // fim da função displayMessage
38 private:
39    string courseName; // nome do curso para esse GradeBook
40 }; // fim da classe GradeBook
41
42 // função main inicia a execução do programa
43 int main()
44 {
45    // cria dois objetos GradeBook
46    GradeBook gradeBook1( "CS101 Introdução à programação C++" );
47    GradeBook gradeBook2( "CS102 Estrutura de dados em C++" );
48
49    // exibe valor inicial de courseName para cada GradeBook
50    cout << "gradeBook1 criado para o curso: " << gradeBook1.getCourseName()
51       << "\ngradeBook2 criado para o curso: " << gradeBook2.getCourseName()
52       << endl;
53 } // fim do main
```

```
gradeBook1 criado para o curso: CS101 Introdução à programação C++
gradeBook2 criado para o curso: CS102 Estruturas de dados em C++
```

Figura 16.7 ■ Instanciando vários objetos da classe GradeBook usando o construtor de GradeBook para especificar o nome do curso quando cada objeto GradeBook for criado. (Parte 2 de 2.)

Review

- A class is formed by a set of methods and attributes that carry out a specific task
- Each method manipulates the attributes of a class
- Each object has its own attributes
  - Accessed by the methods set and get
- Constructors are called to initialize objects before their usage

Curso de Modelagem e Teste de Software Embarcado Automotivo

Escopo de Classe e Destrutores

Prof. Dr. Giovani Gracioli
giovani@lisha.ufsc.br

Prof. Dr. Antônio Augusto Fröhlich
guto@lisha.ufsc.br

ROTA2030
FUNDEP

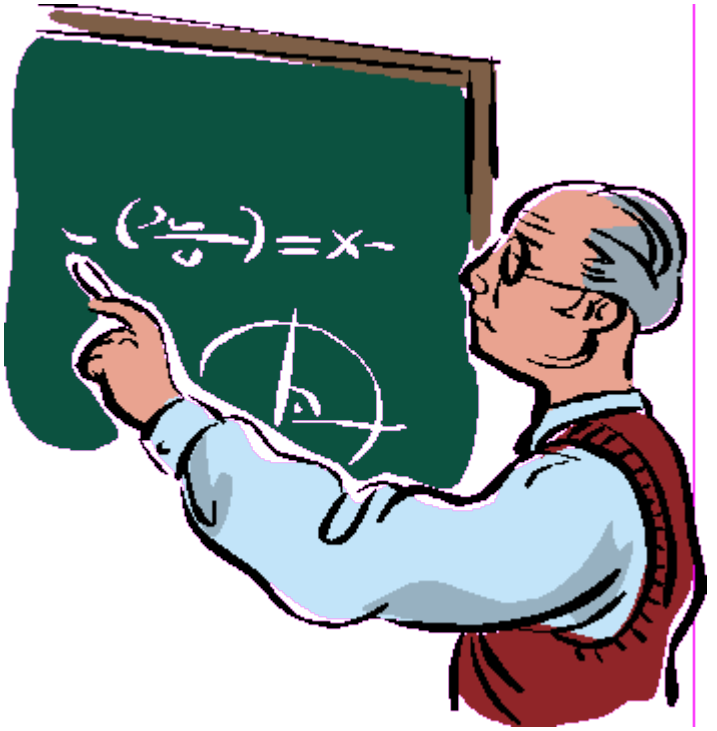
Objectives

- Defining and declaring methods
  - header and implementation files
- Class scope and accessing class members
- Default constructor parameter values
- Destructors
- Default memberwise assignment

What you will learn

- How to separate method definition from its implementation in different files
- How to access an object using its name or a pointer to an object
- How to declare constructors with default arguments
- How to implement destructors
- Default memberwise assignment

Let's get started



Declaring and defining a method

- A method can be declared inside a class, but defined outside this class definition
  - It still keeps the class scope
  - It is only known by other class members
- Method defined inside the class (the way you have learned so far)
  - C++ compiler tries to put all method calls inline into the final code

Example (1)

```
1  // Fig. 9.1: Time.h
2  // Time class definition.
3  // Member functions are defined in Time.cpp
4
5  // prevent multiple inclusions of header
6  #ifndef TIME_H
7  #define TIME_H
8
9  // Time class definition
10 class Time
11 {
12 public:
13    Time(); // constructor
14    void setTime( int, int, int ); // set hour, minute and second
15    void printUniversal(); // print time in universal-time format
16    void printStandard(); // print time in standard-time format
17 private:
18    int hour; // 0 - 23 (24-hour clock format)
19    int minute; // 0 - 59
20    int second; // 0 - 59
21 }; // end class Time
22
23 #endif
```

Example (2)

```
1  // Fig. 9.2: Time.cpp
2  // Member-function definitions for class Time.
3  #include <iostream>
4  #include <iomanip>
5  #include <stdexcept> // for invalid_argument exception class
6  #include "Time.h" // include definition of class Time from Time.h
7
8  using namespace std;
9
10 // Time constructor initializes each data member to zero.
11 Time::Time()
12 {
13    hour = minute = second = 0;
14 } // end Time constructor
```

Class scope and Accessing class member (1)

- The class scope contains
  - Attributes and methods
- Class members can be accessed by all methods
- Outside the class scope
  - Class members defined as public are referenced by a handle
    - name of an object
    - reference to an object
    - pointer to an object
- Methods can be overloaded

Class scope and Accessing class member (2)

- Dot member selection operator (.)
  - used with the object's name or by a reference to an object
- Arrow member selection operator (->)
  - used with a pointer to an object

Example (1)

```
1  // Figura 9.4: fig09_04.cpp
2  // Demonstrando os operadores de acesso ao membro de classe com . e ->
3  #include <iostream>
4  using std::cout;
5  using std::endl;
6
7  // definição da classe Count
8  class Count
9  {
10 public: // dados public são perigosos
11    // configura o valor do membro de dados private x
12    void setX( int value )
13    {
14       x = value;
15    } // fim da função setX
16
17    // imprime o valor do membro de dados private x
18    void print()
19    {
20       cout << x << endl;
21    } // fim da função print
22
23 private:
24    int x;
25 }; // fim da classe Count
```

Example (2)

Usando o operador de seleção de membro ponto com um objeto.

```
26
27 int main()
28 {
29    Count counter; // cria objeto counter
30    Count *counterPtr = &counter; // cria ponteiro para counter
31    Count &counterRef = counter; // criar referência para counter
32
33    cout << "Set x to 1 and print using the object's name: ";
34    counter.setX( 1 ); // configura membro de dados x como 1
35    counter.print(); // chama função-membro print
36
37    cout << "Set x to 2 and print using a reference to an object: ";
38    counterRef.setX( 2 ); // configura membro de dados x como 2
39    counterRef.print(); // chama função-membro print
40
41    cout << "Set x to 3 and print using a pointer to an object: ";
42    counterPtr->setX( 3 ); // configura membro de dados x como 3
43    counterPtr->print(); // chama função-membro print
44    return 0;
45 } // fim de main
```

Usando o operador de seleção de membro ponto com uma referência.

Usando o operador de seleção de membro seta com um ponteiro.

```
Set x to 1 and print using the object's name: 1
Set x to 2 and print using a reference to an object: 2
Set x to 3 and print using a pointer to an object: 3
```

Constructor with default arguments

- The constructors can specify default arguments
  - They can initialize data members in a consistent state
  - even if no values are passed to the method

Example (1)

```
1  // Figura 9.8: Time.h
2  // Declaração da classe Time.
3  // Funções-membro definidas em Time.cpp.
4
5  // impede múltiplas inclusões de arquivo de cabeçalho
6  #ifndef TIME_H
7  #define TIME_H
8
9  // Definição de tipo de dados abstrato Time
10 class Time
11 {
12 public:
13    Time( int = 0, int = 0, int = 0 ); // construtor-padrão
14
15    // funções set
16    void setTime( int, int, int ); // configura hour, minute, second
17    void setHour( int ); // configura hour (depois da validação)
18    void setMinute( int ); // configura minute (depois da validação)
19    void setSecond( int ); // configura second (depois da validação)
```

Example (2)

```
1  // Figura 9.9: Time.cpp
2  // Definições de função-membro para a classe Time.
3  #include <iostream>
4  using std::cout;
5
6  #include <iomanip>
7  using std::setfill;
8  using std::setw;
9
10 #include "Time.h" // inclui a definição da classe Time a partir de Time.h
11
12 // Construtor de Time inicializa cada membro de dados como zero;
13 // assegura que os objetos Time iniciem em um estado consistente
14 Time::Time( int hr, int min, int sec )
15 {
16    setTime( hr, min, sec ); // valida e configura time
17 } // fim do construtor de Time
18
19 // configura novo valor de Time utilizando a hora universal; assegura que
20 // os dados permaneçam consistentes configurando valores inválidos como zero
21 void Time::setTime( int h, int m, int s )
22 {
23    setHour( h ); // configura campo private hour
24    setMinute( m ); // configura campo private minute
25    setSecond( s ); // configura campo private second
26 } // fim da função setTime
27
28 // configura valor de hour
29 void Time::setHour( int h )
```

Example (3)

```
1  // Figura 9.10: fig09_10.cpp
2  // Demonstrando um construtor-padrão para a classe Time.
3  #include <iostream>
4  using std::cout;
5  using std::endl;
6
7  #include "Time.h" // inclui a definição da classe Time a partir de Time.h
8
9  int main()
10 {
11    Time t1; // todos os argumentos convertidos para sua configuração-padrão
12    Time t2( 2 ); // hour especificada; minute e second convertidos para o padrão
13    Time t3( 21, 34 ); // hour e minute especificados; second convertido para o padrão
14    Time t4( 12, 25, 42 ); // hour, minute e second especificados
15    Time t5( 27, 74, 99 ); // valores inválidos especificados
16
17    cout << "Constructed with:\n\nt1: all arguments defaulted\n  ";
18    t1.printUniversal(); // 00:00:00
19    cout << "\n  ";
20    t1.printStandard(); // 12:00:00 AM
21
22    cout << "\n\nt2: hour specified; minute and second defaulted\n  ";
23    t2.printUniversal(); // 02:00:00
24    cout << "\n  ";
25    t2.printStandard(); // 2:00:00 AM
```

Destructors (1)

- As a constructor, a destructor is a special method
- Its name begins with the til character (~) followed by the class name
  - Example: ~Time()
- It is implicitly called when the object is destroyed
  - Ex: when a function ends
- The memory used by the destroyed object can be reused

Destructors (2)

- Does not receive nor return any values
- A class can only have one destructor
  - Overloading of destructor is not allowed
- If the programmer does not implement a destructor, the compiler will offer a standard empty destructor

Example (1)

```
1  // Figura 9.11: CreateAndDestroy.h
2  // Definição da classe CreateAndDestroy.
3  // Funções-membro definidas em CreateAndDestroy.cpp.
4  #include <string>
5  using std::string;
6
7  #ifndef CREATE_H
8  #define CREATE_H
9
10 class CreateAndDestroy
11 {
12 public:
13    CreateAndDestroy( int, string ); // construtor
14    ~CreateAndDestroy(); // destrutor
15 private:
16    int objectID; // Número de ID do objeto
17    string message; // mensagem descrevendo o objeto
18 }; // fim da classe CreateAndDestroy
19
20 #endif
```

Protótipo para o destrutor.

Example (2)

```
1  // Figura 9.12: CreateAndDestroy.cpp
2  // Definições de função-membro da classe CreateAndDestroy.
3  #include <iostream>
4  using std::cout;
5  using std::endl;
6
7  #include "CreateAndDestroy.h"// inclui a definição da classe CreateAndDestroy
8
9  // construtor
10 CreateAndDestroy::CreateAndDestroy( int ID, string messageString )
11 {
12    objectID = ID; // configura o número de ID do objeto
13    message = messageString; // configura mensagem descritiva do objeto
14
15    cout << "Object " << objectID << "   constructor runs   "
16       << message << endl;
17 } // fim do construtor CreateAndDestroy
18
19 // destrutor
20 CreateAndDestroy::~CreateAndDestroy()
21 {
22    // gera saída de nova linha para certos objetos; ajuda a legibilidade
23    cout << ( objectID == 1 || objectID == 6 ? "\n" : "" );
24
25    cout << "Object " << objectID << "   destructor runs    "
26       << message << endl;
27 } // fim do destrutor ~CreateAndDestroy
```

Definindo o destrutor da classe.

Example (3)

```
1  // Figura 9.13: fig09_13.cpp
2  // Demonstrando a ordem em que construtores e
3  // destrutores são chamados.
4  #include <iostream>
5  using std::cout;
6  using std::endl;
7
8  #include "CreateAndDestroy.h" // inclui a definição da classe CreateAndDestroy
9
10 void create( void ); // protótipo
11
12 CreateAndDestroy first( 1, "(global before main)" ); // objeto global
13
14 int main()
15 {
```

Objeto criado fora de **`main.`**

Default memberwise assignment

- The assignment operator (=) can be used to assign an object to another object of the same type
- Each attribute of the object on the left of the assignment operator is assigned to the same attribute in the object on the right side
- Caution: If the attribute is a pointer, this can cause serious problems

Example

```
1  // Figura 9.19: fig09_19.cpp
2  // Demonstrando que os objetos de classe podem ser atribuídos
3  // um ao outro utilizando atribuição-padrão de membro a membro.
4  #include <iostream>
5  using std::cout;
6  using std::endl;
7
8  #include "Date.h" // inclui a definição da classe Date a partir de Date.h
9
10 int main()
11 {
12    Date date1( 7, 4, 2004 );
13    Date date2; // date2 assume padrão de 1/1/2000
14
15    cout << "date1 = ";
16    date1.print();
17    cout << "\ndate2 = ";
18    date2.print();
19
20    date2 = date1; // atribuição-padrão de membro a membro
21
22    cout << "\n\nAfter default memberwise assignment, date2 = ";
23    date2.print();
24    cout << endl;
25    return 0;
26 } // fim de main
```

A atribuição membro a membro atribui membros de dados de **`date1`** a **`date2.`**

```
date1 = 7/4/2004
date2 = 1/1/2000

After default memberwise assignment, date2 = 7/4/2004
```

`date2` agora armazena a mesma data como **`date1.`**

Review

- We can declare a method in a header file and define the method in a cc file
  - All methods in a header files will be inlined
- dot operator (.) and arrow (->) operator to access objects
- Constructors with default arguments
  - This can also be applied to any method
- Destructors
- Default memberwise assignment

Curso de Modelagem e Teste de Software Embarcado Automotivo

Herança

Prof. Dr. Giovani Gracioli
giovani@lisha.ufsc.br

Prof. Dr. Antônio Augusto Fröhlich
guto@lisha.ufsc.br

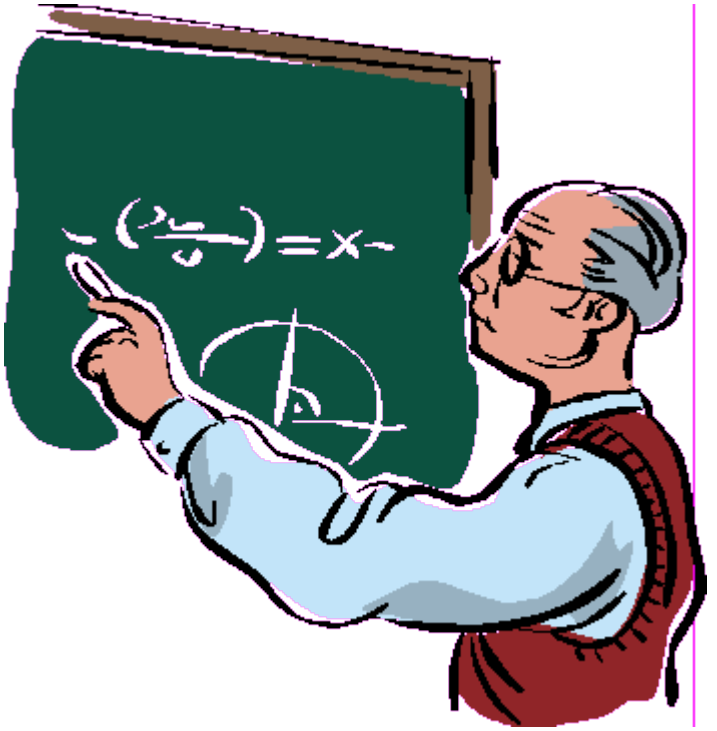
Objectives

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

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

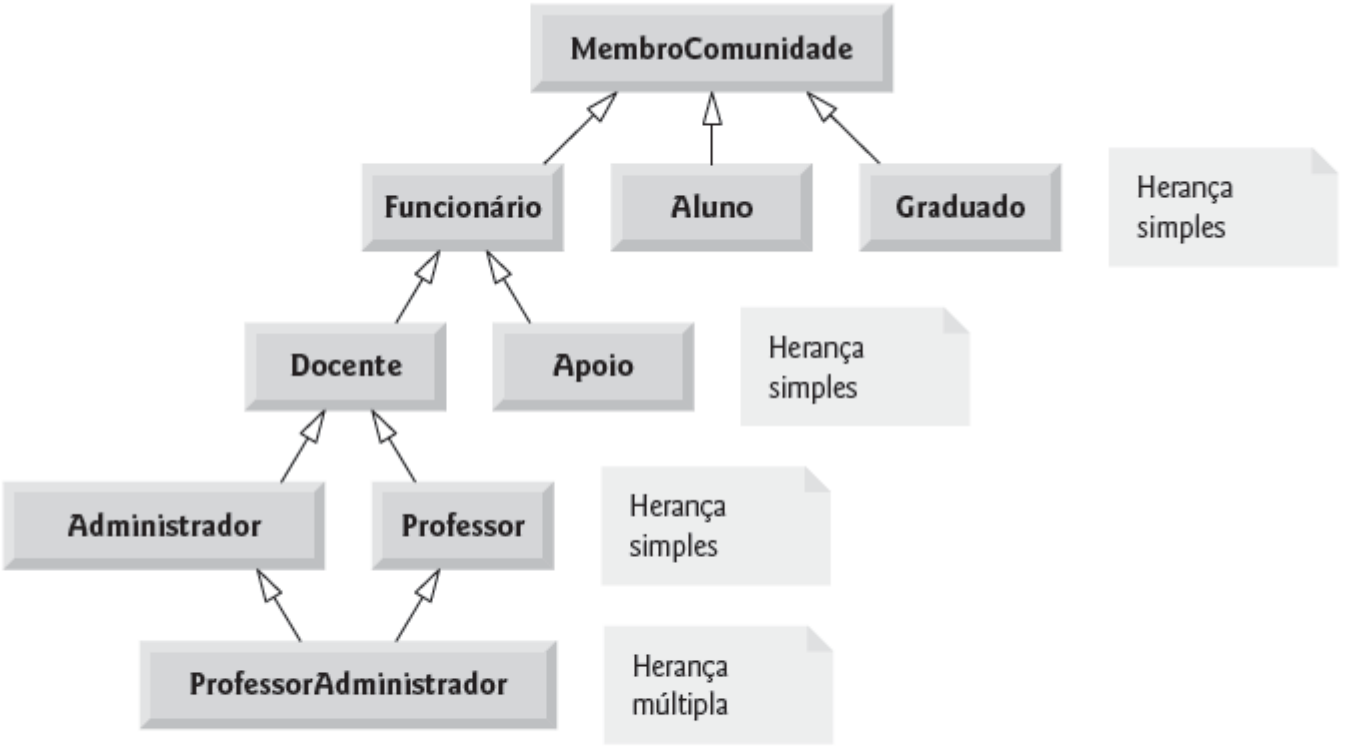


Figura 20.2 ■ Hierarquia de herança de uma universidade `MembroComunidade`.

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

Curso de Modelagem e Teste de Software Embarcado Automotivo

Polimorfismo

Prof. Dr. Giovani Gracioli
giovani@lisha.ufsc.br

Prof. Dr. Antônio Augusto Fröhlich
guto@lisha.ufsc.br

ROTA2030
FUNDEP

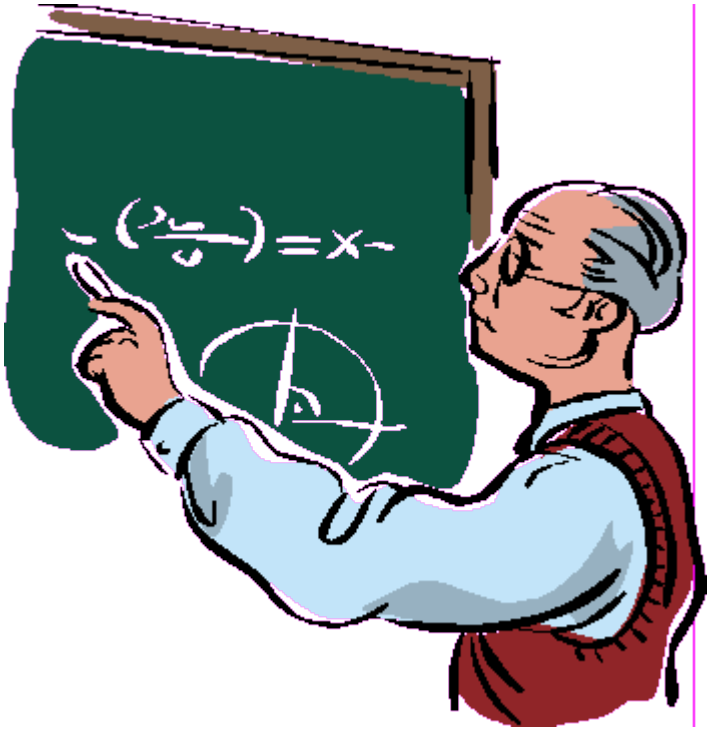
Objectives

- Introduce polymorphism in C++
  - How to declare and use virtual methods
  - How to declare and use abstract classes

What you will learn

- To declare and use virtual methods
- To distinguish between abstract and concrete classes
- To declare pure virtual methods to create abstract classes

Let's get started



Introduction

- Polymorphism enables you to program in the general rather than program in specific
- Enables you to write programs that process objects of classes that are part of the same hierarchy as if they were all objects of the hierarchy's base class
- Design and implement systems that are easily extensible
  - new classes can be added with little or no modification

Example

- Animal hierarchy
  - Base class Animal - all derived class has a move method
  - Different Animal objects are kept as a vector of Animal pointers
  - The program sends the same message (move) for each animal generically
  - The appropriate method is called
    - *Fish* moves by swimming
    - *Frog* moves by jumping
    - *Bird* moves by flying

Main concept

- With public inheritance an object of a derived class can be treated as an object of its base class

- Ex: a program can create an array of base-class pointers that point to objects of many derived-class types
  - Despite the fact that the derived-class objects are different types, the compiler allows this because each derived-class object is an object of its base class

- We cannot treat a base-class object as an object of any of its derived classes

Example

```
1  // Figura 13.1: CommissionEmployee.h
2  // Classe CommissionEmployee representa um empregado comissionado.
3  #ifndef COMMISSION_H
4  #define COMMISSION_H
5
6  #include <string> // classe string padrão C++
7  using std::string;
8
9  class CommissionEmployee
10 {
11 public:
12    CommissionEmployee( const string &, const string &, const string &,
13       double = 0.0, double = 0.0 );
14
15    void setFirstName( const string & ); // configura o nome
16    string getFirstName() const; // retorna o nome
17
18    void setLastName( const string & ); // configura o sobrenome
19    string getLastName() const; // retorna o sobrenome
20
21    void setSocialSecurityNumber( const string & ); // configura o SSN
22    string getSocialSecurityNumber() const; // retorna o SSN
23
24    void setGrossSales( double ); // configura a quantidade de vendas brutas
25    double getGrossSales() const; // retorna a quantidade de vendas brutas
```

Example

A função **`earnings`** será redefinida nas classes derivadas para calcular os rendimentos do funcionário.

```
26
27    void setCommissionRate( double ); // configura a taxa de comissão
28    double getCommissionRate() const; // retorna a taxa de comissão
29
30    double earnings() const; // calcula os rendimentos
31    void print() const; // imprime o objeto CommissionEmployee
32 private:
33    string firstName;
34    string lastName;
35    string socialSecurityNumber;
36    double grossSales; // vendas brutas semanais
37    double commissionRate; // porcentagem da comissão
38 }; // fim da classe CommissionEmployee
39
40 #endif
```

A função **`print`** será redefinida na classe derivada para imprimir informações sobre o funcionário.

Example

```
1  // Figura 13.2: CommissionEmployee.cpp
2  // Definições de função-membro da classe CommissionEmployee.
3  #include <iostream>
4  using std::cout;
5
6  #include "CommissionEmployee.h" // definição da classe CommissionEmployee
7
8  // construtor
9  CommissionEmployee::CommissionEmployee(
10    const string &first, const string &last, const string &ssn,
11    double sales, double rate )
12    : firstName( first ), lastName( last ), socialSecurityNumber( ssn )
13 {
14    setGrossSales( sales ); // valida e armazena as vendas brutas
15    setCommissionRate( rate ); // valida e armazena a taxa de comissão
16 } // fim do construtor CommissionEmployee
17
18 // configura o nome
19 void CommissionEmployee::setFirstName( const string &first )
20 {
21    firstName = first; // deve validar
22 } // fim da função setFirstName
23
24 // retorna o nome
25 string CommissionEmployee::getFirstName() const
26 {
27    return firstName;
28 } // fim da função getFirstName
29
30 // configura o sobrenome
31 void CommissionEmployee::setLastName( const string &last )
```

Example

```
32  {
33     lastName = last;  // deve validar
34  } // fim da função setLastName
35
36  // retorna o sobrenome
37  string CommissionEmployee::getLastName() const
38  {
39     return lastName;
40  } // fim da função getLastName
41
42  // configura o SSN
43  void CommissionEmployee::setSocialSecurityNumber( const string &ssn )
44  {
45     socialSecurityNumber = ssn; // deve validar
46  } // fim da função setSocialSecurityNumber
47
48  // retorna o SSN
49  string CommissionEmployee::getSocialSecurityNumber() const
50  {
51     return socialSecurityNumber;
52  } // fim da função getSocialSecurityNumber
53
54  // configura a quantidade de vendas brutas
55  void CommissionEmployee::setGrossSales( double sales )
56  {
57     grossSales = ( sales < 0.0 ) ? 0.0 : sales;
58  } // fim da função setGrossSales
```

Example

```
59
60  // retorna a quantidade de vendas brutas
61  double CommissionEmployee::getGrossSales() const
62  {
63     return grossSales;
64  } // fim da função getGrossSales
65
66  // configura a taxa de comissão
67  void CommissionEmployee::setCommissionRate( double rate )
68  {
```

Example

```
69     commissionRate = ( rate > 0.0 && rate < 1.0 ) ? rate : 0.0;
70  } // fim da função setCommissionRate
71
72  // retorna a taxa de comissão
73  double CommissionEmployee::getCommissionRate() const
74  {
75     return commissionRate;
76  } // fim da função getCommissionRate
77
78  // calcula os rendimentos
79  double CommissionEmployee::earnings() const
80  {
81     return getCommissionRate() * getGrossSales();
82  } // fim da função earnings
83
84  // imprime o objeto CommissionEmployee
85  void CommissionEmployee::print() const
86  {
87     cout << "commission employee: "
88        << getFirstName() << ' ' << getLastName()
89        << "\nsocial security number: " << getSocialSecurityNumber()
90        << "\ngross sales: " << getGrossSales()
91        << "\ncommission rate: " << getCommissionRate();
92  } // fim da função print
```

Calcula os rendimentos com base na taxa de comissão e nas vendas brutas.

Exibe nome, número de seguro social, vendas brutas e taxa de comissão.

Example

```
1  // Figura 13.3: BasePlusCommissionEmployee.h
2  // Classe BasePlusCommissionEmployee derivada da classe
3  // CommissionEmployee.
4  #ifndef BASEPLUS_H
5  #define BASEPLUS_H
6
7  #include <string> // classe string padrão C++
8  using std::string;
9
10 #include "CommissionEmployee.h" // declaração da classe CommissionEmployee
11
12 class BasePlusCommissionEmployee : public CommissionEmployee
13 {
14 public:
15    BasePlusCommissionEmployee( const string &, const string &,
16       const string &, double = 0.0, double = 0.0, double = 0.0 );
17
18    void setBaseSalary( double ); // configura o salário-base
19    double getBaseSalary() const; // retorna o salário-base
20
21    double earnings() const; // calcula os rendimentos
22    void print() const; // imprime o objeto BasePlusCommissionEmployee
23 private:
24    double baseSalary; // salário-base
25 }; // fim da classe BasePlusCommissionEmployee
26
27 #endif
```

Redefine as funções **`earnings`** e **`print`**.

Example

```
1  // Figura 13.4: BasePlusCommissionEmployee.cpp
2  // Definições de função-membro da classe BasePlusCommissionEmployee.
3  #include <iostream>
4  using std::cout;
5
6  // Definição da classe BasePlusCommissionEmployee
7  #include "BasePlusCommissionEmployee.h"
8
9  // construtor
10 BasePlusCommissionEmployee::BasePlusCommissionEmployee(
11    const string &first, const string &last, const string &ssn,
12    double sales, double rate, double salary )
13    // chama explicitamente o construtor da classe básica
14    : CommissionEmployee( first, last, ssn, sales, rate )
15 {
16    setBaseSalary( salary ); // valida e armazena o salário-base
17 } // fim do construtor BasePlusCommissionEmployee
18
19 // configura o salário-base
20 void BasePlusCommissionEmployee::setBaseSalary( double salary )
21 {
22    baseSalary = ( salary < 0.0 ) ? 0.0 : salary;
23 } // fim da função setBaseSalary
24
25 // retorna o salário-base
26 double BasePlusCommissionEmployee::getBaseSalary() const
27 {
28    return baseSalary;
29 } // fim da função getBaseSalary
```

Example

```
30
31 // calcula os rendimentos
32 double BasePlusCommissionEmployee::earnings() const
33 {
34    return getBaseSalary() + CommissionEmployee::earnings();
35 } // fim da função earnings
36
37 // imprime o objeto BasePlusCommissionEmployee
38 void BasePlusCommissionEmployee::print() const
39 {
40    cout << "base-salaried ";
41
42    // invoca a função print de CommissionEmployee
43    CommissionEmployee::print();
44
45    cout << "\nbase salary: " << getBaseSalary();
46 } // fim da função print
```

A função **`earnings`** redefinida incorpora o salário-base.

A função **`print`** redefinida exibe outros detalhes de **`BasePlusCommissionEmployee`**.

Example: main

```
1  // Figura 13.5: fig13_05.cpp
2  // Apontando ponteiros de classe básica e classe derivada para objetos de classe
3  // básica e classe derivada, respectivamente.
4  #include <iostream>
5  using std::cout;
6  using std::endl;
7  using std::fixed;
8
9  #include <iomanip>
10 using std::setprecision;
11
12 // inclui definições de classe
13 #include "CommissionEmployee.h"
14 #include "BasePlusCommissionEmployee.h"
15
16 int main()
17 {
18    // cria objeto de classe básica
19    CommissionEmployee commissionEmployee(
20       "Sue", "Jones", "222-22-2222", 10000, .06 );
21
22    // cria ponteiro de classe básica
23    CommissionEmployee *commissionEmployeePtr = 0;
24
25    // cria objeto de classe derivada
26    BasePlusCommissionEmployee basePlusCommissionEmployee(
27       "Bob", "Lewis", "333-33-3333", 5000, .04, 300 );
28
29    // cria ponteiro de classe derivada
30    BasePlusCommissionEmployee *basePlusCommissionEmployeePtr = 0;
```

Example: main

```
31
32    // configura a formatação de saída de ponto flutuante
33    cout << fixed << setprecision( 2 );
34
35    // gera saída dos objetos commissionEmployee e basePlusCommissionEmployee
36    cout << "Print base-class and derived-class objects:\n\n";
37    commissionEmployee.print(); // invoca print da classe básica
38    cout << "\n\n";
39    basePlusCommissionEmployee.print(); // invoca print da classe derivada
40
41    // aponta o ponteiro de classe básica para o objeto de classe básica e imprime
42    commissionEmployeePtr = &commissionEmployee; // perfeitamente natural
43    cout << "\n\n\nCalling print with base-class pointer to "
44       << "\nbase-class object invokes base-class print function:\n\n";
45    commissionEmployeePtr->print(); // invoca print da classe básica
```

Direcionando o ponteiro da classe básica para um objeto da classe básica e invocando a funcionalidade da classe básica.

Example: main

```
46
47    // aponta o ponteiro de classe derivada para o objeto de classe derivada e imprime
48    basePlusCommissionEmployeePtr = &basePlusCommissionEmployee; // natural
49    cout << "\n\n\nCalling print with derived-class pointer to "
50       << "\nderived-class object invokes derived-class "
51       << "print function:\n\n";
52    basePlusCommissionEmployeePtr->print(); // invoca print da classe derivada
53
54    // aponta ponteiro de classe básica para o objeto de classe derivada e imprime
55    commissionEmployeePtr = &basePlusCommissionEmployee;
56    cout << "\n\n\nCalling print with base-class pointer to "
57       << "derived-class object\ninvokes base-class print "
58       << "function on that derived-class object:\n\n";
59    commissionEmployeePtr->print(); // invoca print da classe básica
60    cout << endl;
61    return 0;
62 } // fim de main
```

Direcionando o ponteiro da classe básica para um objeto da classe derivada e invocando a funcionalidade da classe básica.

Direcionando o ponteiro da classe derivada para um objeto da classe derivada e invocando a funcionalidade da classe derivada.

Example: main

```
Print base-class and derived-class objects:

commission employee: Sue Jones
social security number: 222-22-2222
gross sales: 10000.00
commission rate: 0.06

base-salaried commission employee: Bob Lewis
social security number: 333-33-3333
gross sales: 5000.00
commission rate: 0.04
base salary: 300.00


Calling print with base-class pointer to
base-class object invokes base-class print function:

commission employee: Sue Jones
social security number: 222-22-2222
gross sales: 10000.00
commission rate: 0.06
```

Example: main

```
Calling print with derived-class pointer to
derived-class object invokes derived-class print function:

base-salaried commission employee: Bob Lewis
social security number: 333-33-3333
gross sales: 5000.00
commission rate: 0.04
base salary: 300.00


Calling print with base-class pointer to derived-class object
invokes base-class print function on that derived-class object:

commission employee: Bob Lewis
social security number: 333-33-3333
gross sales: 5000.00
commission rate: 0.04
```

Aiming derived-class pointers at base-class objects

```
1  // Figura 13.6: fig13_06.cpp
2  // Apontando um ponteiro de classe derivada para um objeto de classe básica.
3  #include "CommissionEmployee.h"
4  #include "BasePlusCommissionEmployee.h"
5
6  int main()
7  {
8     CommissionEmployee commissionEmployee(
9        "Sue", "Jones", "222-22-2222", 10000, .06 );
10    BasePlusCommissionEmployee *basePlusCommissionEmployeePtr = 0;
11
12    // aponta o ponteiro de classe derivada para objeto de classe básica
13    // Erro: um CommissionEmployee não é um BasePlusCommissionEmployee
14    basePlusCommissionEmployeePtr = &commissionEmployee;
15    return 0;
16 } // fim de main
```

Não é possível atribuir objetos da classe básica a um ponteiro da classe derivada porque o relacionamento *é um* não é aplicável.

Mensagens de erro do compilador de linha de comando Borland C++:

```
Error E2034 Fig13_06\fig13_06.cpp 14: Cannot convert 'CommissionEmployee *'
   to 'BasePlusCommissionEmployee *' in function main()
```

Mensagens de erro do compilador GNU C++:

```
fig13_06.cpp:14: error: invalid conversion from `CommissionEmployee*' to
   `BasePlusCommissionEmployee*'
```

Mensagens de erro do compilador Microsoft Visual C++.NET:

```
C:\cpphtp5_examples\ch13\Fig13_06\fig13_06.cpp(14) : error C2440:
   '=' : cannot convert from 'CommissionEmployee *__w64 ' to
   'BasePlusCommissionEmployee *'
        Cast from base to derived requires dynamic_cast or static_cast
```

Derived-class method calls via base-class pointers

```
1  // Figura 13.7: fig13_07.cpp
2  // Tentando invocar as funções-membro exclusivas da classe derivada
3  // por um ponteiro de classe básica.
4  #include "CommissionEmployee.h"
5  #include "BasePlusCommissionEmployee.h"
6
7  int main()
8  {
9     CommissionEmployee *commissionEmployeePtr = 0; // classe básica
10    BasePlusCommissionEmployee basePlusCommissionEmployee(
11       "Bob", "Lewis", "333-33-3333", 5000, .04, 300 ); // classe derivada
12
13    // aponta o ponteiro de classe básica para o objeto de classe derivada
14    commissionEmployeePtr = &basePlusCommissionEmployee;
15
16    // invoca as funções-membro de classe básica no objeto de classe derivada
17    // por ponteiro de classe básica
18    string firstName = commissionEmployeePtr->getFirstName();
19    string lastName = commissionEmployeePtr->getLastName();
20    string ssn = commissionEmployeePtr->getSocialSecurityNumber();
21    double grossSales = commissionEmployeePtr->getGrossSales();
22    double commissionRate = commissionEmployeePtr->getCommissionRate();
23
24    // tentativa de invocar funções exclusivas de classe derivada
25    // em objeto de classe derivada por meio de um ponteiro de classe básica
26    double baseSalary = commissionEmployeePtr->getBaseSalary();
27    commissionEmployeePtr->setBaseSalary( 500 );
28      return 0;
29 } // fim de main
```

Não é possível invocar membros apenas da classe derivada a partir do ponteiro da classe básica.

Derived-class method calls via base-class pointers

Mensagens de erro do compilador de linha de comando Borland C++:

```
Error E2316 Fig13_07\fig13_07.cpp 26: 'getBaseSalary' is not a member of
   'CommissionEmployee' in function main()
Error E2316 Fig13_07\fig13_07.cpp 27: 'setBaseSalary' is not a member of
   'CommissionEmployee' in function main()
```

Mensagens de erro do compilador Microsoft Visual C++.NET:

```
C:\cpphtp5_examples\ch13\Fig13_07\fig13_07.cpp(26) : error C2039:
   'getBaseSalary' : is not a member of 'CommissionEmployee'
      C:\cpphtp5_examples\ch13\Fig13_07\CommissionEmployee.h(10) :
         see declaration of 'CommissionEmployee'
C:\cpphtp5_examples\ch13\Fig13_07\fig13_07.cpp(27) : error C2039:
   'setBaseSalary' : is not a member of 'CommissionEmployee'
      C:\cpphtp5_examples\ch13\Fig13_07\CommissionEmployee.h(10) :
         see declaration of 'CommissionEmployee'
```

Mensagens de erro do compilador GNU C++:

```
fig13_07.cpp:26: error: `getBaseSalary' undeclared (first use this function)
fig13_07.cpp:26: error: (Each undeclared identifier is reported only once for
   each function it appears in.)
fig13_07.cpp:27: error: `setBaseSalary' undeclared (first use this function)
```

Virtual Functions

- Recall that the type of the handle determines which class's functionality to invoke
  - Ex: A base-class pointer will call the method of the base-class and not the method of the derived-class, even if it pointers to an object of the derived-class type
- With virtual functions, the type of the object, not the type of the handle used to invoke the member function, determines which version of a virtual function to invoke

Virtual Functions

- Allows the program to dynamically determine (at run-time) which method should be called
- Suppose that shape classes such as Circle, Triangle, Rectangle, ans Square, are all derived from base class Shape
  - They have a different method draw
  - In a program that draws a set of shapes, it would be useful to be able to treat all shapes generically as objects of the base class Shape
  - A base-class Shape pointer to invoke draw based on the type of the object to which the base-class Shape pointer points at any given time

Virtual Functions

- To enable this behavior, we declare draw in the base class as a virtual function, and we override draw in each of the derived classes to draw the appropriate shape

- Example of how to declare

```
virtual void draw();
```

Example

```
1  // Figura 13.8: CommissionEmployee.h
2  // Classe CommissionEmployee representa um empregado comissionado.
3  #ifndef COMMISSION_H
4  #define COMMISSION_H
5
6  #include <string> // classe string padrão C++
7  using std::string;
8
9  class CommissionEmployee
10 {
11 public:
12    CommissionEmployee( const string &, const string &, const string &,
13       double = 0.0, double = 0.0 );
14
15    void setFirstName( const string & ); // configura o nome
16    string getFirstName() const; // retorna o nome
17
18    void setLastName( const string & ); // configura o sobrenome
19    string getLastName() const; // retorna o sobrenome
20
21    void setSocialSecurityNumber( const string & ); // configura o SSN
22    string getSocialSecurityNumber() const; // retorna o SSN
23
24    void setGrossSales( double ); // configura a quantidade de vendas brutas
25    double getGrossSales() const; // retorna a quantidade de vendas brutas
```

Example

```
26
27     void setCommissionRate( double ); // configura a taxa de comissão
28     double getCommissionRate() const; // retorna a taxa de comissão
29
30     virtual double earnings() const; // calcula os rendimentos
31     virtual void print() const; // imprime o objeto CommissionEmployee
32  private:
33     string firstName;
34     string lastName;
35     string socialSecurityNumber;
36     double grossSales; // vendas brutas semanais
37     double commissionRate; // porcentagem da comissão
38  }; // fim da classe CommissionEmployee
39
40  #endif
```

Declarar **`earnings`** e **`print`** como **`virtual`** permite que elas sejam sobrescritas, mas não redefinidas.

Example

```
1  // Figura 13.9: BasePlusCommissionEmployee.h
2  // Classe BasePlusCommissionEmployee derivada da classe
3  // CommissionEmployee.
4  #ifndef BASEPLUS_H
5  #define BASEPLUS_H
6
7  #include <string> // classe string padrão C++
8  using std::string;
9
10 #include "CommissionEmployee.h" // declaração da classe CommissionEmployee
11
12 class BasePlusCommissionEmployee : public CommissionEmployee
13 {
14 public:
15    BasePlusCommissionEmployee( const string &, const string &,
16       const string &, double = 0.0, double = 0.0, double = 0.0 );
17
18    void setBaseSalary( double ); // configura o salário-base
19    double getBaseSalary() const; // retorna o salário-base
20
21    virtual double earnings() const; // calcula os rendimentos
22    virtual void print() const; // imprime o objeto BasePlusCommissionEmployee
23 private:
24    double baseSalary; // salário-base
25 }; // fim da classe BasePlusCommissionEmployee
26
27 #endif
```

As funções **`earnings`** e **`print`** continuam sendo **`virtual`** — é sempre bom declarar **`virtual`** mesmo ao sobrescrever uma função.

Example: main

```
1  // Figura 13.10: fig13_10.cpp
2  // Introduzindo polimorfismo, funções virtual e vinculação dinâmica.
3  #include <iostream>
4  using std::cout;
5  using std::endl;
6  using std::fixed;
7
8  #include <iomanip>
9  using std::setprecision;
10
11 // inclui definições de classe
12 #include "CommissionEmployee.h"
13 #include "BasePlusCommissionEmployee.h"
14
15 int main()
16 {
17    // cria objeto de classe básica
18    CommissionEmployee commissionEmployee(
19       "Sue", "Jones", "222-22-2222", 10000, .06 );
20
21    // cria ponteiro de classe básica
22    CommissionEmployee *commissionEmployeePtr = 0;
23
24    // cria objeto de classe derivada
25    BasePlusCommissionEmployee basePlusCommissionEmployee(
26       "Bob", "Lewis", "333-33-3333", 5000, .04, 300 );
27
28    // cria ponteiro de classe derivada
29    BasePlusCommissionEmployee *basePlusCommissionEmployeePtr = 0;
30
```

Example: main

```
31    // configura a formatação de saída de ponto flutuante
32    cout << fixed << setprecision( 2 );
33
34    // gera saída de objetos utilizando vinculação estática
35    cout << "Invoking print function on base-class and derived-class "
36       << "\nobjects with static binding\n\n";
37    commissionEmployee.print(); // vinculação estática
38    cout << "\n\n";
39    basePlusCommissionEmployee.print(); // vinculação estática
40
41    // gera saída de objetos utilizando vinculação dinâmica
42    cout << "\n\n\nInvoking print function on base-class and "
43       << "derived-class \nobjects with dynamic binding";
44
45    // aponta o ponteiro de classe básica para o objeto de classe básica e imprime
46    commissionEmployeePtr = &commissionEmployee;
47    cout << "\n\nCalling virtual function print with base-class pointer"
48       << "\nto base-class object invokes base-class "
49       << "print function:\n\n";
50    commissionEmployeePtr->print(); // invoca print da classe básica
```

Direcionando o ponteiro da classe básica para um objeto da classe básica e invocando a funcionalidade da classe básica.

Example: main

```
51
52     // aponta o ponteiro de classe derivada para o objeto de classe derivada e imprime
53     basePlusCommissionEmployeePtr = &basePlusCommissionEmployee;
54     cout << "\n\nCalling virtual function print with derived-class "
55        << "pointer\nto derived-class object invokes derived-class "
56        << "print function:\n\n";
57     basePlusCommissionEmployeePtr->print(); // invoca print da classe derivada
58
59     // aponta o ponteiro de classe básica para o objeto de classe derivada e imprime
60     commissionEmployeePtr = &basePlusCommissionEmployee;
61     cout << "\n\nCalling virtual function print with base-class pointer"
62        << "\nto derived-class object invokes derived-class "
63        << "print function:\n\n";
64
65     // polimorfismo; invoca print de BasePlusCommissionEmployee;
66     // ponteiro de classe básica para objeto de classe derivada
67     commissionEmployeePtr->print();
68     cout << endl;
69     return 0;
70  } // fim de main
```

Direcionando o ponteiro da classe derivada para um objeto da classe derivada e invocando a funcionalidade da classe derivada.

Apontando o ponteiro da classe básica para um objeto da classe derivada e invocando a funcionalidade da classe derivada por meio de polimorfismo e das funções **`virtual`**.

Example: main

```
Invoking print function on base-class and derived-class
objects with static binding

commission employee: Sue Jones
social security number: 222-22-2222
gross sales: 10000.00
commission rate: 0.06

base-salaried commission employee: Bob Lewis
social security number: 333-33-3333
gross sales: 5000.00
commission rate: 0.04
base salary: 300.00


Invoking print function on base-class and derived-class
objects with dynamic binding

Calling virtual function print with base-class pointer
to base-class object invokes base-class print function:

commission employee: Sue Jones
social security number: 222-22-2222
gross sales: 10000.00
commission rate: 0.06

Calling virtual function print with derived-class pointer
to derived-class object invokes derived-class print function:

base-salaried commission employee: Bob Lewis
```

Example: main

```
social security number: 333-33-3333
gross sales: 5000.00
commission rate: 0.04
base salary: 300.00

Calling virtual function print with base-class pointer
to derived-class object invokes derived-class print function:

base-salaried commission employee: Bob Lewis
social security number: 333-33-3333
gross sales: 5000.00
commission rate: 0.04
base salary: 300.00
```

Abstract classes and pure virtual functions

- Abstract classes are classes that cannot be instantiated
- Usually, are used as base classes in inheritance hierarchies
  - Abstract base classes
- Abstract classes are incomplete – derived classes must define the missing pieces
- Classes that can be used to instantiate objects are called concrete classes

Pure Virtual Functions

- A class is made abstract by declaring one or more of its virtual functions to be “pure”
- A pure virtual function is specified by placing “= 0” in its declaration
  - virtual void draw() = 0;
- The “= 0” is a pure specifier

Pure Virtual Functions

- The difference between a virtual function and a pure virtual function is that a virtual function has an implementation and gives the derived class the option of overriding the function;

- by contrast, a pure virtual function does not provide an implementation and requires the derived class to override the function for that derived class to be concrete; otherwise the derived class remains abstract

Pure Virtual Functions

- Are used when it does not make sense for the base class to have an implementation of a function, but you want all concrete derived classes to implement the function
- We cannot instantiate objects of an abstract base class, but we can use the abstract base class to declare pointers and references that can refer to objects of any concrete classes derived from the abstract class

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.