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

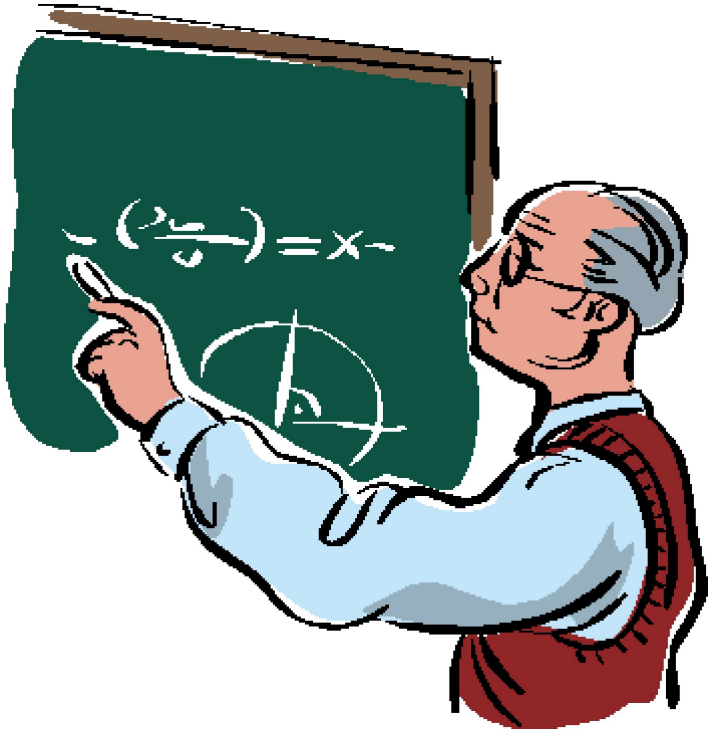
What you need to know to follow

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

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

Tasks

- Read chapter 13 of the text book (C++ how to program 8th edition)

- Exercises of chapter 13
  - 13.3, 13.6, 13.8, 13.10, 13.12, 13.13, 13.15

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.