

Programação Orientada a Objetos para Redes de Computadores

Prof. Miguel Elias Mitre Campista

<http://www.gta.ufrj.br/~miguel>

POO para Redes de Computadores - COPPE-PEE/UFRJ

Prof. Miguel Campista

PARTE 2

Programação em C++ - Polimorfismo

POO para Redes de Computadores - COPPE-PEE/UFRJ

Prof. Miguel Campista

Polimorfismo com Hierarquias de Herança

- Polimorfismo permite:
 - "Programar no geral" ao invés de "programar no específico"
 - Processar objetos de classes da mesma hierarquia
 - Como se fossem todos objetos da classe base
- Cada objeto executa as tarefas que lhe são pertinentes
 - Diferentes ações ocorrem, dependendo do tipo de objeto
- Novas classes podem ser adicionadas
 - Com pouca ou nenhuma modificação no código existente

POO para Redes de Computadores - COPPE-PEE/UFRJ

Prof. Miguel Campista

Exemplo: Hierarquia Animal

- Classe base Animal
 - Toda classe derivada tem a função `move`
- Diferentes objetos Animal são mantidos como um vector de ponteiros Animal
- O programa emite a mesma mensagem (`move`) a cada animal genericamente
- A função apropriada é chamada
 - Peixe movimenta-se (`move`) nadando
 - Sapo movimenta-se (`move`) pulando
 - Pássaro movimenta-se (`move`) voando

POO para Redes de Computadores - COPPE-PEE/UFRJ

Prof. Miguel Campista

Exemplos de Polimorfismo

- O polimorfismo ocorre quando um programa invoca uma função virtual por meio de um ponteiro ou referência de classe base
 - C++ escolhe dinamicamente a função correta para a classe na qual o objeto foi instanciado

POO para Redes de Computadores - COPPE-PEE/UFRJ

Prof. Miguel Campista

Observações de Engenharia de Software

- Funções virtual combinadas com o polimorfismo permitem tratar de generalidades e deixar as questões específicas dos objetos para o ambiente em tempo de execução
 - É possível direcionar uma variedade de objetos a se comportar de maneira apropriada sem mesmo conhecer seus tipos
 - Contudo que esses objetos pertençam à mesma hierarquia de herança e estejam sendo acessados por meio de um ponteiro de classe base comum

POO para Redes de Computadores - COPPE-PEE/UFRJ

Prof. Miguel Campista

Observações de Engenharia de Software

- O polimorfismo promove extensibilidade
 - O software escrito para invocar comportamento polimórfico é escrito independentemente dos tipos de objeto para os quais as mensagens são enviadas
 - É possível incorporar nesse sistema novos tipos de objeto que podem responder a mensagens existentes sem modificar o sistema de base
 - Somente o código do cliente que instancia os novos objetos deve ser modificado para acomodar os novos tipos

POO para Redes de Computadores - COPPE-PEE/UFRJ

Prof. Miguel Campista

Relacionamento de Objetos em Hierarquia de Herança

- Conceito-chave
 - Um objeto de uma classe derivada pode ser tratado como um objeto de sua classe base
 - Objeto da classe derivada "é um" objeto da classe base

POO para Redes de Computadores - COPPE-PEE/UFRJ

Prof. Miguel Campista

Uso de Fçs. de Classe Base de Obj's de Classe Derivada

- Apontar um ponteiro de classe **base** para um objeto de classe **base**
 - Invoca a funcionalidade da classe base
- Apontar um ponteiro de classe **derivada** para um objeto de classe **derivada**
 - Invoca a funcionalidade da classe derivada

POO para Redes de Computadores - COPPE-PEE/UFRJ

Prof. Miguel Campista

Uso de Fçs. de Classe Base de Obj's de Classe Derivada

- Apontar um ponteiro de classe **base** para um objeto de classe **derivada**
 - Objeto de classe derivada é um objeto de classe base
 - Chama a funcionalidade da classe base
 - A funcionalidade invocada depende do tipo do ponteiro ou referência (*handle*) usado para invocar a função, não do tipo de objeto para o qual o *handle* aponta

```
commissionEmployeePtr = &basePlusCommissionEmployee;  
// Chama a fç print declarada na classe base  
commissionEmployeePtr->print ();
```

POO para Redes de Computadores - COPPE-PEE/UFRJ

Prof. Miguel Campista

Uso de Fçs. de Classe Base de Obj's de Classe Derivada

- Apontar um ponteiro de classe **base** para um objeto de classe **derivada**
 - Funções virtual
 - Permitem que se invoque a funcionalidade do tipo de objeto, ao invés da funcionalidade do tipo de *handle*
 - São fundamentais para implementar comportamento polimórfico

POO para Redes de Computadores - COPPE-PEE/UFRJ

Prof. Miguel Campista

```
/*  
 * Aula 13 - Exemplo 1  
 * Arquivo comissionCap13Ex1.h  
 * Autor: Miguel Campista  
 */  
#ifndef COMMISSION_H  
#define COMMISSION_H  
  
#include <iostream>  
#include <string>  
#include <iomanip>  
  
using namespace std;  
  
class CommissionEmployee {  
public:  
    CommissionEmployee (const string &, const string &,  
                        const string &, double = 0.0, double = 0.0);  
  
    void setFirstName (const string &); // Configura o nome  
    string getFirstName () const; // Retorna o nome  
  
    void setLastName (const string &);  
    string getLastName () const;  
  
    void setSocialSecurityNumber (const string &);  
    string getSocialSecurityNumber () const;  
  
    void setGrossSales (double); // Configura o quant. de vendas brutas  
    double getGrossSales () const; // Retorna a quant. de vendas brutas  
  
    void setCommissionRate (double); // Conf. taxa de comissão  
    double getCommissionRate () const;  
  
    double earnings () const; // Calcula os rendimentos  
    void print () const;
```

Primeiro Exemplo de Polimorfismo em C++

```
private:
    string firstName, lastName;
    string socialSecurityNumber;
    double grossSales;
    double commissionRate;
};

#endif
```

POO para Redes de Computadores - COPPE-PEE/UFRJ

Prof. Miguel Campista

```
/*
 * Aula 13 - Exemplo 1
 * Arquivo commissionCap13Ex1.cpp
 * Autor: Miguel Campista
 */
#include "commissionCap13Ex1.h"

CommissionEmployee::CommissionEmployee (const string &first, const string &last,
    const string &ssn, double sales, double rate) :
    firstName (first), lastName (last), socialSecurityNumber (ssn) {
    setGrossSales (sales);
    setCommissionRate (rate);
}

void CommissionEmployee::setFirstName (const string &first) {
    firstName = first;
}

string CommissionEmployee::getFirstName () const { return firstName; }

void CommissionEmployee::setLastName (const string &last) {
    lastName = last;
}

string CommissionEmployee::getLastName () const { return lastName; }

void CommissionEmployee::setSocialSecurityNumber (const string &ssn) {
    socialSecurityNumber = ssn;
}

string CommissionEmployee::getSocialSecurityNumber () const {
    return socialSecurityNumber;
}

void CommissionEmployee::setGrossSales (double sales) {
    grossSales = (sales < 0.0) ? 0.0 : sales;
}
```

Primeiro Exemplo de Polimorfismo em C++

```
double CommissionEmployee::getGrossSales () const { return grossSales; }

void CommissionEmployee::setCommissionRate (double rate) {
    commissionRate = (rate > 0.0 && rate < 1.0) ? rate : 0.0;
}

double CommissionEmployee::getCommissionRate () const { return commissionRate; }

double CommissionEmployee::earnings () const {
    return getGrossSales () * getCommissionRate ();
}

void CommissionEmployee::print () const {
    cout << "Commission employee: " << getFirstName ()
    << " " << getLastName ()
    << "\nsocial security number: " << getSocialSecurityNumber ()
    << "\ngross sales: " << getGrossSales ()
    << "\ncommission rate: " << getCommissionRate ();
}
```

POO para Redes de Computadores - COPPE-PEE/UFRJ

Prof. Miguel Campista

Primeiro Exemplo de Polimorfismo em C++

```
/*
 * Aula 13 - Exemplo 1
 * Arquivo basePlusCommissionCap13Ex1.h
 * Autor: Miguel Campista
 */
#ifndef BASEPLUS_H
#define BASEPLUS_H

#include <iostream>
#include <string>
#include <iomanip>
#include "commissionCap13Ex1.h"

using namespace std;

class BasePlusCommissionEmployee : public CommissionEmployee {
public:
    BasePlusCommissionEmployee (const string &, const string &,
        const string &, double = 0.0, double = 0.0, double = 0.0);

    void setBaseSalary (double); // Conf. o salário base
    double getBaseSalary () const;

    double earnings () const; // Calcula os rendimentos
    void print () const;
private:
    double baseSalary;
};

#endif
```

Primeiro Exemplo de Polimorfismo em C++

```
/*
 * Aula 13 - Exemplo 1
 * Arquivo basePlusCommissionCap13Ex1.cpp
 * Autor: Miguel Campista
 */
#include "basePlusCommissionCap13Ex1.h"

BasePlusCommissionEmployee::BasePlusCommissionEmployee (const string &first,
    const string &last, const string &ssn,
    double sales, double rate, double salary) :
    CommissionEmployee (first, last, ssn, sales, rate) {
    setBaseSalary (salary);
}

void BasePlusCommissionEmployee::setBaseSalary (double salary) {
    baseSalary = (salary < 0.0) ? 0.0 : salary;
}

double BasePlusCommissionEmployee::getBaseSalary () const { return baseSalary; }

double BasePlusCommissionEmployee::earnings () const {
    return baseSalary + CommissionEmployee::earnings ();
}

void BasePlusCommissionEmployee::print () const {
    cout << "Base salary: " << endl;
    CommissionEmployee::print ();
    cout << "\nbase salary: " << getBaseSalary ();
}
```

Primeiro Exemplo de Polimorfismo em C++

```
/*
 * Aula 13 - Exemplo 1
 * Programa Principal
 * Autor: Miguel Campista
 */
#include "basePlusCommissionCap13Ex1.h"

int main () {
    // Cria um objeto de classe básica
    CommissionEmployee * commissionEmployeePtr = NULL;
    commissionEmployeePtr = new CommissionEmployee ("Sue", "Jones",
        "21-1111-1111", 10000, .06);

    // Cria um ponteiro de classe básica
    CommissionEmployee * commissionEmployeePtr = NULL;

    // Cria um objeto de classe derivada
    BasePlusCommissionEmployee * basePlusCommissionEmployeePtr = new BasePlusCommissionEmployee ("Bob", "Lewis",
        "21-2222-2222", 5000, .04, 300);

    // Cria um ponteiro de classe derivada
    BasePlusCommissionEmployee * basePlusCommissionEmployeePtr = NULL;

    // Configura a formatação de saída de ponto flutuante
    cout << fixed << setprecision (2);

    // Gere a saída dos objetos commissionEmployee
    // e basePlusCommissionEmployee
    cout << "Imprima os objetos da classe base e da derivada:\n\n";
    commissionEmployeePtr->print (); // Invoca print da classe base
    cout << "\n\n";
    basePlusCommissionEmployeePtr->print (); // Invoca print da classe derivada
}
```

Primeiro Exemplo de Polimorfismo em C++

```
// Aponta o ponteiro de classe base para
// o objeto de classe base e imprime
commissionEmployeePtr = &commissionEmployee; // perfeitamente natural
cout << "\n\nChamando print com o ponteiro da classe base para "
<< "objeto da classe base invoca funcao print da classe base!\n\n";
commissionEmployeePtr->print (); // Invoca print da classe base

// Aponta o ponteiro de classe derivada para
// objeto de classe derivada e imprime
basePlusCommissionEmployeePtr = &basePlusCommissionEmployee; // natural
cout << "\n\nChamando print com o ponteiro da classe derivada para "
<< "objeto da classe derivada invoca funcao print da "
<< "classe derivada!\n\n";
basePlusCommissionEmployeePtr->print (); // Invoca print da cl. derivada

// Aponta o ponteiro de classe base para
// o objeto de classe derivada e imprime
commissionEmployeePtr = &basePlusCommissionEmployee; // natural
cout << "\n\nChamando print com o ponteiro da classe base para "
<< "objeto da classe derivada invoca funcao print da "
<< "classe base no objeto da classe derivada!\n\n";
commissionEmployeePtr->print (); // Invoca print da classe base
cout << endl;

return 0;
```

1

Primeiro Exemplo de Polimorfismo em C++

```
// Aponta o
// o objeto
// o objeto de classe derivada e da derivada:
commissionEmployeePtr = &commissionEmployee; // perfeitamente natural
cout << "\n\nBase salary
<< "Objeto de classe derivada invoca funcao print da classe base!
commissionEmployeePtr->print (); // Invoca print da classe base

// Aponta o
// o objeto de classe derivada e da derivada:
basePlusCommissionEmployeePtr = &basePlusCommissionEmployee; // natural
cout << "\n\nChamando print com o ponteiro da classe base para objeto da classe base invoca f
basePlusCommissionEmployeePtr->print (); // Invoca print da classe base

// Aponta o
// o objeto de classe derivada e da derivada:
commissionEmployeePtr = &commissionEmployee; // natural
cout << "\n\nChamando print com o ponteiro da classe base para objeto da classe derivada
basePlusCommissionEmployeePtr->print (); // Invoca print da classe derivada

// Aponta o
// o objeto de classe derivada e da derivada:
basePlusCommissionEmployeePtr = &basePlusCommissionEmployee; // natural
cout << "\n\nChamando print com o ponteiro da classe base para objeto da classe derivada
basePlusCommissionEmployeePtr->print (); // Invoca print da classe derivada

return 0;
```

1

Ponteiro de Classe Derivada p/ um Objeto de Classe Base

- Apontar um ponteiro de classe derivada para um objeto de classe base
 - Faz com que o compilador C++ gere erros
 - CommissionEmployee (objeto da classe base) não é um BasePlusCommissionEmployee (objeto da classe derivada)
 - Se isso fosse permitido, o programador poderia tentar acessar membros de dados da classe derivada que não existem na classe base
 - Por exemplo, ele poderia tentar acessar baseSalary
 - Como não existe, a modificação dessa área de memória pode sobrescrever a memória em uso por outros dados

POO para Redes de Computadores - COPPE-PEE/UFRJ

Prof. Miguel Campista

Segundo Exemplo de Polimorfismo em C++

```
/*
 * Aula 13 - Exemplo 2
 * Programa Principal
 * Autor: Miguel Campista
 */
#include "CommissionCap13Ex1.h"
#include "basePlusCommissionCap13Ex1.h"

int main () {
    // Cria um objeto de classe base
    CommissionEmployee commissionEmployee ("Sue", "Jones",
                                             "21-1111-1111", 10000, .06);

    BasePlusCommissionEmployee *basePlusCommissionEmployeePtr = 0;

    // Aponta o ponteiro de classe derivada para objeto de classe base
    // Erro: um CommissionEmployee não é um BasePlusCommissionEmployee
    basePlusCommissionEmployeePtr = &commissionEmployee;

    return 0;
}
```

POO para Redes de Computadores - COPPE-PEE/UFRJ

Prof. Miguel Campista

Segundo Exemplo de Polimorfismo em C++

```
//
 * Aula 13 - Exemplo 2
 * Programa Principal
 * Autor: Miguel Campista
 */
Line File Message
18 C:\Users\Miguel\Documents\UFRJ... In function int main():
C:\Users\Miguel\Documents\UFRJ... invalid conversion from 'CommissionEmployee' to 'BasePlusCommissionEmployee'
C:\Users\Miguel\Documents\UFRJ... [-fpermissive] [Bad Error: = 16413202] [Error]

// Cria um objeto de classe base
CommissionEmployee commissionEmployee ("Sue", "Jones",
                                         "21-1111-1111", 10000, .06);

BasePlusCommissionEmployee *basePlusCommissionEmployeePtr = 0;

// Aponta o ponteiro de classe derivada para objeto de classe base
// Erro: um CommissionEmployee não é um BasePlusCommissionEmployee
basePlusCommissionEmployeePtr = &commissionEmployee;

return 0;

}
```

POO para Redes de Computadores - COPPE-PEE/UFRJ

Prof. Miguel Campista

Uso de Fçs. de Cl. Derivada via Ponteiros de Cl. Base

- Apontando um ponteiro de classe base para um objeto de classe derivada
 - Chamar funções existentes na classe base faz com que a funcionalidade da classe base seja invocada
 - Chamar funções não existentes na classe base (mas que podem existir na classe derivada) provocará erros
 - Os membros da classe derivada não podem ser acessados por meio de ponteiros de classe base
 - Entretanto, podem ser executados por meio de **downcasting**

POO para Redes de Computadores - COPPE-PEE/UFRJ

Prof. Miguel Campista

Terceiro Exemplo de Polimorfismo em C++

```

/*
 * Aula 13 - Exemplo 3
 * Programa Principal
 * Autor: Miguel Campista
 */
#include "ComissaoCapit3Ex1.h"
#include "basePlusComissaoCapit3Ex1.h"

int main () {
    // Classe base
    ComissaoEmployee *comissaoEmployeePtr = 0;

    // Classe derivada
    basePlusComissaoEmployee basePlusComissaoEmployee(
        "Bob", "Lewis", "333-33-3333", 5000, .04, 300 );

    // Aponta o ponteiro de classe base para o objeto da classe derivada
    comissaoEmployeePtr = &basePlusComissaoEmployee;

    // Invoca as funções membro de classe base no objeto de classe derivada
    // por ponteiro de classe base
    string firstName = comissaoEmployeePtr->getFirstName();
    string lastName = comissaoEmployeePtr->getLastName();
    string emp = comissaoEmployeePtr->getSocialSecurityNumber();
    double grossSales = comissaoEmployeePtr->getGrossSales();
    double commissionRate = comissaoEmployeePtr->getCommissionRate();

    // Tentativa de invocar funções exclusivas de classe derivada
    // no objeto de classe derivada por meio de um ponteiro de classe base
    double baseSalary = comissaoEmployeePtr->getBaseSalary();
    comissaoEmployeePtr->setBaseSalary( 500 );

    return 0;
}

```

Terceiro Exemplo de Polimorfismo em C++

```

/*
 * Aula 13 - Exemplo 3
 * Programa Principal
 * Autor: Miguel Campista
 */
#include "ComissaoCapit3Ex1.h"
#include "basePlusComissaoCapit3Ex1.h"

int main () {
    // Classe base
    ComissaoEmployee *comissaoEmployeePtr = 0;

    // Classe derivada
    basePlusComissaoEmployee basePlusComissaoEmployee(
        "Bob", "Lewis", "333-33-3333", 5000, .04, 300 );

    // Aponta o ponteiro de classe base para o objeto da classe derivada
    comissaoEmployeePtr = &basePlusComissaoEmployee;

    // Invoca as funções membro de classe base no objeto de classe derivada
    // por ponteiro de classe base
    string firstName = comissaoEmployeePtr->getFirstName();
    string lastName = comissaoEmployeePtr->getLastName();
    string emp = comissaoEmployeePtr->getSocialSecurityNumber();
    double grossSales = comissaoEmployeePtr->getGrossSales();
    double commissionRate = comissaoEmployeePtr->getCommissionRate();

    // Tentativa de invocar funções exclusivas de classe derivada
    // no objeto de classe derivada por meio de um ponteiro de classe base
    double baseSalary = comissaoEmployeePtr->getBaseSalary();
    comissaoEmployeePtr->setBaseSalary( 500 );

    return 0;
}

```

Line	File	Message
30	C:\Users\Miguel\Documents\UFRJ\...	In function 'int main()': 'class ComissaoEmployee' has no member named 'getBaseSalary'
31	C:\Users\Miguel\Documents\UFRJ\...	'class ComissaoEmployee' has no member named 'setBaseSalary'
	C:\Users\Miguel\Documents\UFRJ\...	[Build Error] exe: "" [aula13-ex3.o] Error 1

Exemplo 1

- Escreva um programa que invoque o método `move` comum em três classes derivadas da classe `Animal` e imprima uma mensagem. Para isso, crie uma classe `Agua`, `Cavalo` e `Tubarao`. A função principal terá um array de ponteiros para objetos da classe `Animal` cujos elementos serão endereços de objetos das classes derivadas.



Exemplo 1

```

#include <iostream>
#include "CavaloCapit3Ex8.h"
#include "TubaraoCapit3Ex8.h"
#include "aguaCapit3Ex8.h"

using namespace std;

int main () {
    Animal *a [3];

    a [0] = new Cavalo;
    a [1] = new Tubarao;
    a [2] = new Agua;

    cout << endl;

    a [0]->move ();
    a [1]->move ();
    a [2]->move ();

    delete a [0];
    delete a [1];
    delete a [2];

    return 0;
}

```

Exemplo 1

```

#include <iostream>
using namespace std;

#include "animal.h"

void animal::move () {
    cout << "O animal moveu!" << endl;
}

class Animal {
public:
    void move ();
};

#endif

```

Exemplo 1

```

#include <iostream>
#include "animal.h"

using namespace std;

#include "agua.h"

Agua::Agua () {
    cout << "Nasceu a agua!" << endl;
}

void Agua::move () {
    cout << "A agua voou!" << endl;
}

class Agua : public Animal {
public:
    Agua ();
    void move ();
};

#endif

```

Exemplo 1

```
#include <iostream>
#include "animal.h"
using namespace std;
#define TUBARAO_H
#define TUBARAO_H
class Tubarao : public Animal {
public:
    Tubarao ();
    void move ();
};
#endif
```

```
#include "tubarao.h"
Tubarao::Tubarao () {
    cout << "Nasceu o tubarao!" << endl;
}
void Tubarao::move () {
    cout << "O tubarao nadou!" << endl;
}
```

POO para Redes de Computadores - COPPE-PEE/UFRJ

Prof. Miguel Campista

Exemplo 1

```
#include <iostream>
#include "animal.h"
using namespace std;
#define CAVALO_H
#define CAVALO_H
class Cavalo : public Animal {
public:
    Cavalo ();
    void move ();
};
#endif
```

```
#include "cavalo.h"
Cavalo::Cavalo () {
    cout << "Nasceu o cavalo!" << endl;
}
void Cavalo::move () {
    cout << "O cavalo galopou!" << endl;
}
```

POO para Redes de Computadores - COPPE-PEE/UFRJ

Prof. Miguel Campista

Exemplo 1

```
#include <iostream>
#include "animal.h"
using namespace std;
#define CAVALO_H
#define CAVALO_H
class Cavalo : public Animal {
public:
    Cavalo ();
    void move ();
};
#endif
```

POO para Redes de Computadores - COPPE-PEE/UFRJ

Prof. Miguel Campista

Exemplo 1

```
#include <iostream>
#include "animal.h"
using namespace std;
#define CAVALO_H
#define CAVALO_H
class Cavalo : public Animal {
public:
    Cavalo ();
    void move ();
};
#endif
```

E se a gente criar um ponteiro para um objeto da classe Cavalo que recebe o conteúdo do array na posição 0?

POO para Redes de Computadores - COPPE-PEE/UFRJ

Prof. Miguel Campista

Exemplo 1

```
#include <iostream>
#include "cavaloCap13Ex8.h"
#include "tubaraoCap13Ex8.h"
#include "aguiaCap13Ex8.h"
using namespace std;
int main () {
    Animal *a [3];
    a [0] = new Cavalo;
    a [1] = new Tubarao;
    a [2] = new Aguia;
    cout << endl;
    a [0]->move ();
    a [1]->move ();
    a [2]->move ();
    Cavalo *c = a [0];
    delete a [0];
    delete a [1];
    delete a [2];
}
```

Funciona?

POO para Redes de Computadores - COPPE-PEE/UFRJ

Prof. Miguel Campista

Exemplo 1

```
#include <iostream>
#include "cavaloCap13Ex8.h"
#include "tubaraoCap13Ex8.h"
#include "aguiaCap13Ex8.h"
using namespace std;
int main () {
    Animal *a [3];
    a [0] = new Cavalo;
    a [1] = new Tubarao;
    a [2] = new Aguia;
    cout << endl;
    a [0]->move ();
    a [1]->move ();
    a [2]->move ();
    Cavalo *c = a [0];
    delete a [0];
    delete a [1];
    delete a [2];
    return 0;
}
```

21 C:\Users\Miguel\Documents\UFRJ\... In function 'int main()':
C:\Users\Miguel\Documents\UFRJ\... invalid conversion from 'Animal*' to 'Cavalo*'
C:\Users\Miguel\Documents\UFRJ\... [Build Error] exe: *** [principalCap13Ex8.o] Error 1

POO para Redes de Computadores - COPPE-PEE/UFRJ

Prof. Miguel Campista

Uso de Fçs. de Cl. Derivada via Ponteiros de Cl. Base

- Se o endereço de um objeto de classe derivada foi atribuído a um ponteiro de uma de suas classes base diretas ou indiretas, é aceitável fazer coerção desse ponteiro de classe base de volta para um ponteiro de classe derivada
 - Na verdade, isso deve ser feito para enviar a esse objeto de classe derivada mensagens que não aparecem na classe base
 - Mas para isso a classe base deve ser **POLIMÓRFICA!**

Como criar classes polimórficas?

POO para Redes de Computadores - COPPE-PEE/UFRJ

Prof. Miguel Campista

Funções Virtuais

- Qual função da classe deve ser invocada?
 - Normalmente
 - O *handle* determina a funcionalidade da classe a ser invocada
 - Com funções `virtual`
 - O tipo do objeto para o qual está se apontando, não o tipo do *handle*, determina que versão da função `virtual` deve ser invocada
 - Programa determina dinamicamente (em tempo de execução, e não em tempo de compilação) que função deve ser usada
 - Procedimento conhecido por vinculação dinâmica ou vinculação tardia (*late binding*)

POO para Redes de Computadores - COPPE-PEE/UFRJ

Prof. Miguel Campista

Funções Virtuais

- São declaradas colocando-se antes do protótipo de função a palavra-chave `virtual` na classe base
- As classes derivadas sobrescrevem uma função `virtual` quando apropriado
- Depois que é declarada `virtual`...
 - A função permanece `virtual` em todos os níveis inferiores da hierarquia

POO para Redes de Computadores - COPPE-PEE/UFRJ

Prof. Miguel Campista

Funções Virtuais

- Vinculação estática
 - Ao chamar uma função `virtual` usando um objeto específico com o operador ponto, a invocação da função é resolvida em tempo de compilação
- Vinculação dinâmica
 - A vinculação dinâmica ocorre somente com os *handles* de ponteiro e de referência

```
CommissionEmployee c; //Classe derivada
c.print ();
```

```
CommissionEmployee ce; //Classe derivada
Employee *c = &ce;
c->print ();
```

POO para Redes de Computadores - COPPE-PEE/UFRJ

Prof. Miguel Campista

Funções Virtuais

- Uma vez que uma função é declarada `virtual`, permanece `virtual` por toda a hierarquia de herança a partir desse ponto
 - Mesmo que essa função não seja declarada explicitamente como `virtual` quando uma classe a sobrescreve
- Mesmo que uma função seja implicitamente `virtual` por causa de uma declaração feita em um ponto mais alto da hierarquia de classes
 - Declare explicitamente essa função `virtual` em cada nível da hierarquia para deixar o programa mais claro

POO para Redes de Computadores - COPPE-PEE/UFRJ

Prof. Miguel Campista

Funções Virtuais

- Quando uma classe derivada escolhe não sobrescrever uma função `virtual` de sua classe base...
 - A classe derivada simplesmente herda a implementação de função `virtual` de sua classe base

POO para Redes de Computadores - COPPE-PEE/UFRJ

Prof. Miguel Campista

```

/*
 * Aula 13 - Exemplo 4
 * Arquivo comissionCap3Ex4.h
 * Autor: Miguel Campista
 */
#ifndef COMMISSION_H
#define COMMISSION_H
#include <iostream>
#include <string>
#include <iomanip>

using namespace std;

class CommissionEmployee {
public:
    CommissionEmployee (const string &, const string &,
                       const string &, double = 0.0, double = 0.0);

    void setFirstName (const string &); // Configura o nome
    string getFirstName () const; // Retorna o nome

    void setLastName (const string &);
    string getLastName () const;

    void setSocialSecurityNumber (const string &);
    string getSocialSecurityNumber () const;

    void setGrossSales (double); // Configura a quant. de vendas brutas
    double getGrossSales () const; // Retorna a quant. de vendas brutas

    void setCommissionRate (double); // Conf. taxa de comissão
    double getCommissionRate () const;

    virtual double earnings () const; // Calcula os rendimentos
    virtual void print () const;
};

```

Quarto Exemplo de Polimorfismo em C++

```

private:
    string firstName, lastName;
    string socialSecurityNumber;
    double grossSales;
    double commissionRate;
};

#endif

```

POO para Redes de Computadores - COPPE-PEE/UF RJ

Prof. Miguel Campista

Quarto Exemplo de Polimorfismo em C++

```

/*
 * Aula 13 - Exemplo 4
 * Arquivo comissionCap3Ex4.cpp
 * Autor: Miguel Campista
 */
#include "comissionCap3Ex4.h"

CommissionEmployee::CommissionEmployee (const string &first, const string &last,
                                        const string &ssn, double sales, double rate) {
    firstName = first;
    lastName = last;
    socialSecurityNumber = ssn;
    setGrossSales (sales);
    setCommissionRate (rate);
}

void CommissionEmployee::setFirstName (const string &first) {
    firstName = first;
}

string CommissionEmployee::getFirstName () const { return firstName; }

void CommissionEmployee::setLastName (const string &last) {
    lastName = last;
}

string CommissionEmployee::getLastName () const { return lastName; }

void CommissionEmployee::setSocialSecurityNumber (const string &ssn) {
    socialSecurityNumber = ssn;
}

string CommissionEmployee::getSocialSecurityNumber () const {
    return socialSecurityNumber;
}

```

Quarto Exemplo de Polimorfismo em C++

```

void CommissionEmployee::setGrossSales (double sales) {
    grossSales = (sales < 0.0) ? 0.0 : sales;
}

double CommissionEmployee::getGrossSales () const { return grossSales; }

void CommissionEmployee::setCommissionRate (double rate) {
    commissionRate = (rate > 0.0 && rate < 1.0) ? rate : 0.0;
}

double CommissionEmployee::getCommissionRate () const { return commissionRate; }

double CommissionEmployee::earnings () const {
    return getGrossSales () * getCommissionRate ();
}

void CommissionEmployee::print () const {
    cout << "comission employee: " << getFirstName ()
         << " " << getLastName ()
         << "\nsocial security number: " << getSocialSecurityNumber ()
         << "\ngross sales: " << getGrossSales ()
         << "\ncommission rate: " << getCommissionRate ();
}

```

POO para Redes de Computadores - COPPE-PEE/UF RJ

Prof. Miguel Campista

Quarto Exemplo de Polimorfismo em C++

```

/*
 * Aula 13 - Exemplo 4
 * Arquivo BasePlusCommissionCap3Ex4.h
 * Autor: Miguel Campista
 */
#ifndef BASEPLUS_H
#define BASEPLUS_H
#include <iostream>
#include <string>
#include <iomanip>
#include "comissionCap3Ex4.h"

using namespace std;

class BasePlusCommissionEmployee : public CommissionEmployee {
public:
    BasePlusCommissionEmployee (const string &, const string &,
                                const string &, double = 0.0, double = 0.0, double = 0.0);

    void setBaseSalary (double); // Conf. o salário base
    double getBaseSalary () const;

    virtual double earnings () const; // Calcula os rendimentos
    virtual void print () const;

private:
    double baseSalary;
};

#endif

```

Quarto Exemplo de Polimorfismo em C++

```

/*
 * Aula 13 - Exemplo 4
 * Arquivo BasePlusCommissionCap3Ex4.cpp
 * Autor: Miguel Campista
 */
#include "BasePlusCommissionCap3Ex4.h"

BasePlusCommissionEmployee::BasePlusCommissionEmployee (const string &first,
                                                         const string &last, const string &ssn,
                                                         double sales, double rate, double salary) :
    CommissionEmployee (first, last, ssn, sales, rate) {
    setBaseSalary (salary);
}

void BasePlusCommissionEmployee::setBaseSalary (double salary) {
    baseSalary = (salary < 0.0) ? 0.0 : salary;
}

double BasePlusCommissionEmployee::getBaseSalary () const { return baseSalary; }

double BasePlusCommissionEmployee::earnings () const {
    return baseSalary + CommissionEmployee::earnings ();
}

void BasePlusCommissionEmployee::print () const {
    cout << "Base salary" << endl;
    CommissionEmployee::print ();

    cout << "\nbase salary: " << getBaseSalary ();
}

```


Quarto Exemplo de Polimorfismo em C++

```

/*
 * Aula 13 - Exemplo 4
 * Programa Principal
 * Autor: Miguel Campista
 */
#include "basePlusCommissionCap13Ex4.h"

int main ()
// Cria um objeto de classe básica
CommissionEmployee commissionEmployee ("Bua", "Jones",
                                         "21-1111-1111", 10000, .06);

// Cria um ponteiro de classe básica
CommissionEmployee * commissionEmployeePtr = NULL;

// Cria um objeto de classe derivada
BasePlusCommissionEmployee basePlusCommissionEmployee ("Bob", "Lewis",
                                                         "21-2222-2222", 5000, .04, 3000);

// Cria um ponteiro de classe derivada
BasePlusCommissionEmployee * basePlusCommissionEmployeePtr = NULL;

// Configura a formatação de saída de ponto flutuante
cout << fixed << setprecision (2);

// Gera a saída dos objetos commissionEmployee
// e basePlusCommissionEmployee
cout << "Imprime os objetos de classe base e da derivada:\n\n";
commissionEmployee.print (); // Invoca print de classe base
cout << "\n\n";
basePlusCommissionEmployee.print (); // Invoca print de classe derivada

```

Quarto Exemplo de Polimorfismo em C++

```

// Aponta o ponteiro de classe básica para
// o objeto de classe básica e imprime
commissionEmployeePtr = &commissionEmployee; // perfeitamente natural
cout << "\n\nChamando print com o ponteiro da classe base para "
     << "objeto de classe base invoca função print de classe base:\n\n";
commissionEmployeePtr->print (); // Invoca print de classe base

// Aponta o ponteiro de classe derivada para
// o objeto de classe derivada e imprime
basePlusCommissionEmployeePtr = &basePlusCommissionEmployee; // natural
cout << "\n\nChamando print com o ponteiro da classe derivada para "
     << "objeto de classe derivada invoca função print da "
     << "classe derivada:\n\n";
basePlusCommissionEmployeePtr->print (); // Invoca print da cl. derivada

// Aponta o ponteiro de classe base para
// o objeto de classe derivada e imprime
commissionEmployeePtr = &basePlusCommissionEmployee; // natural
cout << "\n\nChamando print com o ponteiro de classe base para "
     << "objeto de classe derivada invoca função print da "
     << "classe derivada no objeto de classe derivada:\n\n";
commissionEmployeePtr->print ();
cout << endl;

return 0;
}

```

Quarto Exemplo de Polimorfismo em C++

```

C:\Users\Miguel\Documents\UFRRJ\disciplinas\linguagem\projeto\aula13-ex4.exe
Imprime os objetos de classe base e da derivada:
// 1
commission employee: Sue Jones
social security number: 21-1111-1111
gross salary: 10000.00
commission rate: 0.06
cout
base salary
commission employee: Bob Lewis
social security number: 21-2222-2222
gross salary: 5000.00
commission rate: 0.04
base salary: 3000.00
// 2
// 3
Chamando print com o ponteiro de classe base para objeto de classe base invoca f
unção print de classe base:
commission employee: Sue Jones
social security number: 21-1111-1111
commission rate: 0.06
base
// 4
Chamando print com o ponteiro de classe derivada para objeto de classe derivada
invoca função print de classe derivada:
base salary
commission employee: Bob Lewis
social security number: 21-2222-2222
gross salary: 5000.00
commission rate: 0.04
base salary: 3000.00
cout
// 5
Chamando print com o ponteiro de classe base para objeto de classe derivada
invoca função print de classe derivada no objeto de classe derivada:
base salary
commission employee: Bob Lewis
social security number: 21-2222-2222
gross salary: 5000.00
commission rate: 0.04
base salary: 3000.00
Pressione qualquer tecla para continuar. . .

```

Atribuições entre Obj's de Cl. Base/Derivada e Ponteiros

- Quatro maneiras de apontar ponteiros de classe base e classe derivada para objetos de classe base e classe derivada
 - Apontar um ponteiro de classe base para um objeto de classe base
 - É um processo direto
 - Apontar um ponteiro de classe derivada para um objeto de classe derivada
 - É um processo direto

Atribuições entre Obj's de Cl. Base/Derivada e Ponteiros

- Quatro maneiras de apontar ponteiros de classe base e classe derivada para objetos de classe base e classe derivada
 - Apontar um ponteiro de classe base para um objeto de classe derivada
 - É seguro, mas pode ser usado para invocar apenas funções-membro que a classe base declara (a menos que seja utilizado downcasting)
 - Permite polimorfismo com funções virtual
 - Apontar um ponteiro de classe derivada para um objeto de classe base
 - Gera um erro de compilação

Atribuições entre Obj's de Cl. Base/Derivada e Ponteiros

- É um erro de compilação...
 - Depois de apontar um ponteiro de classe base para um objeto de classe derivada, tentar referenciar membros exclusivos de classe derivada com o ponteiro da classe base
- Tratar um objeto de classe base como um objeto de classe derivada pode causar erros

Classes Abstratas e Funções Virtual Puras

- Classes abstratas
 - Classes das quais o programador não pretende instanciar objetos
 - São **incompletas**
 - As classes derivadas têm que definir as "partes ausentes"
 - São muito genéricas para definir objetos

POO para Redes de Computadores - COPPE-PEE/UFRJ

Prof. Miguel Campista

Classes Abstratas e Funções Virtual Puras

- Classes abstratas
 - São normalmente usadas como classes base, denominadas classes base abstratas
 - Oferecem uma classe base apropriada da qual outras classes podem herdar
 - As classes usadas para instanciar objetos são denominadas **classes concretas**
 - Devem fornecer implementação a toda função-membro da classe, inclusive as incompletas herdadas

POO para Redes de Computadores - COPPE-PEE/UFRJ

Prof. Miguel Campista

Classes Abstratas e Funções Virtual Puras

- Função **virtual pura**
 - Para tornar uma classe abstrata, é necessário declarar "pura" uma ou mais de suas funções **virtual**
 - Colocando "**= 0**" em sua declaração
 - Ex: `virtual void draw() const = 0;`
 - » "**= 0**" é conhecido como um **especificador puro**

POO para Redes de Computadores - COPPE-PEE/UFRJ

Prof. Miguel Campista

Classes Abstratas e Funções Virtual Puras

- Função **virtual pura**
 - Não fornece implementações
 - Toda classe derivada concreta deve **sobrescrever todas as funções virtual puras de classe base com implementações concretas**
 - Se não sobrescrever, a classe derivada também será abstrata
 - É usada enquanto ainda não fizer sentido para a classe base ter a implementação da função
 - O programador quer que todas as classes derivadas concretas implementem essa função para o seu caso específico

POO para Redes de Computadores - COPPE-PEE/UFRJ

Prof. Miguel Campista

Classes Abstratas e Funções Virtual Puras

- Uma classe abstrata define uma interface pública comum para as várias classes em uma hierarquia de classes
 - Uma classe abstrata contém uma ou mais funções **virtual puras** que as classes derivadas concretas devem sobrescrever
- Tentar instanciar um objeto de uma classe abstrata causa um erro de compilação

POO para Redes de Computadores - COPPE-PEE/UFRJ

Prof. Miguel Campista

Classes Abstratas e Funções Virtual Puras

- A falha em sobrescrever uma função **virtual pura** em uma classe derivada e então tentar instanciar objetos dessa classe é um erro de compilação
- Uma classe abstrata tem pelo menos uma função **virtual pura**
 - Uma classe abstrata também pode ter membros de dados e funções concretas (incluindo construtores e destrutores), que estão sujeitos às regras normais de herança por classes derivadas

POO para Redes de Computadores - COPPE-PEE/UFRJ

Prof. Miguel Campista

Classes Abstratas e Funções Virtual Puras

- Podemos usar uma classe base abstrata para declarar ponteiros e referências
 - É possível fazer referência a todos os objetos de qualquer classe concreta derivada da classe abstrata
- ```
Derivada d; //Obj. da classe derivada
Base *b = &d; //Ponteiro da classe base
```
- Os programas normalmente usam esses ponteiros e referências para manipular objetos de classes derivadas polimorficamente

POO para Redes de Computadores - COPPE-PEE/UFRJ

Prof. Miguel Campista

## Sistema de Folha de Pagamento com Polimorfismo

- Aperfeiçoe a hierarquia `CommissionEmployee-BasePlusCommissionEmployee` usando uma classe base abstrata
  - A classe abstrata `Employee` representa o conceito geral de um empregado
    - Declara a "interface" à hierarquia
    - Todo empregado tem nome, sobrenome e um número de seguro social
  - Os rendimentos são calculados diferentemente e os objetos são impressos diferentemente para cada classe derivada

POO para Redes de Computadores - COPPE-PEE/UFRJ

Prof. Miguel Campista

## Criação da Classe Base Abstrata Employee

- Classe `Employee`
  - Oferece várias funções `get` e `set`
  - Oferece as funções `earnings` e `print`
    - A função `earnings` depende do tipo de empregado, de modo que é declarada **virtual pura**
      - Não há informações suficientes na classe `Employee` para uma implementação-padrão
    - A função `print` é **virtual, mas não virtual pura**
      - A implementação-padrão é fornecida em `Employee`
  - O exemplo mantém um vector de ponteiros `Employee`
    - As funções `earnings` e `print` apropriadas são invocadas polimorficamente

POO para Redes de Computadores - COPPE-PEE/UFRJ

Prof. Miguel Campista

```
/*
 * Aula 13 - Exemplo 6
 * Arquivo comissionCap13Ex6.h
 * Autor: Miguel Campista
 */
#ifndef EMPLOYEE_H
#define EMPLOYEE_H

#include <iostream>
#include <string>
#include <map>

using namespace std;

class Employee {
public:
 Employee (const string &, const string &, const string &);

 void setFirstName (const string &); // Configura o nome
 string getFirstName () const; // Retorna o nome

 void setLastName (const string &);
 string getLastName () const;

 void setSocialSecurityNumber (const string &);
 string getSocialSecurityNumber () const;

 // A função virtual pura cria a classe básica abstrata Employee
 virtual double earnings () const = 0; // Virtual Pura
 virtual void print () const; // Virtual

private:
 string firstName, lastName;
 string socialSecurityNumber;
};

#endif
```

```
/*
 * Aula 13 - Exemplo 6
 * Arquivo comissionCap13Ex6.cpp
 * Autor: Miguel Campista
 */
#include "comissionCap13Ex6.h"

Employee::Employee (const string &first, const string &last, const string &ssn)
 : firstName (first), lastName (last), socialSecurityNumber (ssn) {}

void Employee::setFirstName (const string &first) {
 firstName = first;
}

string Employee::getFirstName () const { return firstName; }

void Employee::setLastName (const string &last) {
 lastName = last;
}

string Employee::getLastName () const { return lastName; }

void Employee::setSocialSecurityNumber (const string &ssn) {
 socialSecurityNumber = ssn;
}

string Employee::getSocialSecurityNumber () const {
 return socialSecurityNumber;
}

// Imprime informações de Employee (virtual, mas não virtual pura)
void Employee::print () const {
 cout << getFirstName () << " " << getLastName ()
 << "\nSocial security number: " << getSocialSecurityNumber ();
}
}
```

POO para Redes de Computadores - COPPE-PEE/UFRJ

Prof. Miguel Campista

## Criação da Classe Derivada Concreta SalariedEmployee

- `SalariedEmployee` herda de `Employee`
  - Inclui um salário semanal
    - A função `earnings` sobrescrita incorpora o salário semanal
    - A função `print` sobrescrita incorpora o salário semanal
  - É uma classe concreta
    - Implementa todas as funções **virtual puras** da classe base abstrata

POO para Redes de Computadores - COPPE-PEE/UFRJ

Prof. Miguel Campista

## Quinto Exemplo de Polimorfismo em C++

```

/*
 * Aula 13 - Exemplo 5
 * Arquivo salariedemployeeCap13Ex5.h
 * Autor: Miguel Campista
 */
#ifndef SALARIED_H
#define SALARIED_H

#include "employeeCap13Ex5.h"

class SalariedEmployee : public Employee {
public:
 SalariedEmployee (const string &first, const string &last,
 const string &id, double = 0.0);

 void setWeeklySalary (double); // Conf. o salário semanal
 double getWeeklySalary () const;

 // Palavra-chave virtual assinala a intenção de sobrescrever
 virtual double earnings () const; // Calcula os rendimentos
 virtual void print () const;

private:
 double weeklySalary;
};

#endif

```

POO para Redes de Computadores - COPPE-PEE/UFRJ

Prof. Miguel Campista

## Quinto Exemplo de Polimorfismo em C++

```

/*
 * Aula 13 - Exemplo 5
 * Arquivo basePlusComissaoCap13Ex5.cpp
 * Autor: Miguel Campista
 */
#include "salariedemployeeCap13Ex5.h"

SalariedEmployee::SalariedEmployee (const string &first,
 const string &last, const string &ssn, double salary) :
 Employee (first, last, ssn) {
 setWeeklySalary (salary);
}

void SalariedEmployee::setWeeklySalary (double salary) {
 weeklySalary = (salary < 0.0) ? 0.0 : salary;
}

double SalariedEmployee::getWeeklySalary () const {
 return weeklySalary;
}

// Calcula os rendimentos
// Sobrescreve a fç virtual para earnings em Employee
double SalariedEmployee::earnings () const {
 return getWeeklySalary ();
}

void SalariedEmployee::print () const {
 cout << "Salaried employee: ";
 Employee::print (); // Reutiliza a fç print da classe base abstrata
 cout << "\nweekly salary: " << getWeeklySalary ();
}

```

## Criação da Classe Derivada Concreta HourlyEmployee

- HourlyEmployee herda de Employee
  - Inclui salário-hora e as horas trabalhadas
    - A função earnings sobrescreta incorpora o salário multiplicados pelas horas (leva em conta o pagamento de 50% a mais)
    - A função print sobrescreta incorpora o salário e as horas trabalhadas
  - É uma classe concreta
    - Implementa todas as funções virtual puras na classe base abstrata

POO para Redes de Computadores - COPPE-PEE/UFRJ

Prof. Miguel Campista

## Quinto Exemplo de Polimorfismo em C++

```

/*
 * Aula 13 - Exemplo 5
 * Arquivo hourlyemployeeCap13Ex5.h
 * Autor: Miguel Campista
 */
#ifndef HOURLY_H
#define HOURLY_H

#include "employeeCap13Ex5.h"

class HourlyEmployee : public Employee {
public:
 HourlyEmployee (const string &first, const string &last,
 const string &id, double = 0.0, double = 0.0);

 void setWage (double); // Conf. o salário por hora
 double getWage () const;

 void setHours (double); // Conf. as horas trabalhadas
 double getHours () const;

 // Palavra-chave virtual assinala a intenção de sobrescrever
 virtual double earnings () const; // Calcula os rendimentos
 virtual void print () const; // Imprime os objetos HourlyEmployee

private:
 double wage, hours;
};

#endif

```

## Quinto Exemplo de Polimorfismo em C++

```

/*
 * Aula 13 - Exemplo 5
 * Arquivo hourlyemployeeCap13Ex5.cpp
 * Autor: Miguel Campista
 */
#include "hourlyemployeeCap13Ex5.h"

HourlyEmployee::HourlyEmployee (const string &first, const string &last,
 const string &ssn, double hourlyWage, double hoursWorked) :
 Employee (first, last, ssn) {
 setWage (hourlyWage);
 setHours (hoursWorked);
}

void HourlyEmployee::setWage (double hourlyWage) {
 wage = (hourlyWage < 0.0) ? 0.0 : hourlyWage;
}

double HourlyEmployee::getWage () const { return wage; }

void HourlyEmployee::setHours (double hoursWorked) {
 hours = (((hoursWorked) >= 0.0) && (hoursWorked) <= 168.0) ?
 hoursWorked : 0.0;
}

double HourlyEmployee::getHours () const { return hours; }

```

POO para Redes de Computadores - COPPE-PEE/UFRJ

Prof. Miguel Campista

## Quinto Exemplo de Polimorfismo em C++

```

// Calcula os rendimentos - Sobrescreve a fç virtual para earnings em Employee
double HourlyEmployee::earnings () const {
 if (getHours () <= 40) // nenhuma hora extra
 return getWage () * getHours ();
 else
 return 40 * getWage () + ((getHours () - 40) * getWage () * 1.5);
}

void HourlyEmployee::print () const {
 cout << "Hourly employee: ";
 Employee::print (); // Reutiliza a fç print da classe base abstrata
 cout << "\nhourly wage: " << getWage ()
 << "\nhours worked: " << getHours ();
}

```

POO para Redes de Computadores - COPPE-PEE/UFRJ

Prof. Miguel Campista

## Classe Derivada Concreta CommissionEmployee

- **CommissionEmployee herda de Employee**
  - Inclui vendas brutas e taxa de comissão
    - A função **earnings** sobrescrita incorpora as vendas brutas e a taxa de comissão
    - A função **print** sobrescrita incorpora as vendas brutas e a taxa de comissão
  - Classe concreta
    - Implementa todas as funções **virtual puras** na classe base abstrata

POO para Redes de Computadores - COPPE-PEE/UFRJ

Prof. Miguel Campista

## Quinto Exemplo de Polimorfismo em C++

```

/*
 * Aula 13 - Exemplo 5
 * Arquivo commissionCap13Ex5.h
 * Autor: Miguel Campista
 */
#ifndef COMMISSION_H
#define COMMISSION_H

#include "employeeCap13Ex5.h"

class CommissionEmployee : public Employee {
public:
 CommissionEmployee (const string &, const string &,
 const string &, double = 0.0, double = 0.0);

 void setCommissionRate (double); // Configura a comissão
 double getCommissionRate () const;

 void setGrossSales (double); // Configura a quant. de vendas brutas
 double getGrossSales () const; // Retorna a quant. de vendas brutas

 // Palavra-chave virtual assinala a intenção de sobrescrever
 virtual double earnings () const; // Calcula os rendimentos
 virtual void print () const; // Imprime os objetos HourlyEmployee

private:
 double grossSales;
 double commissionRate;
};

#endif

```

```

/*
 * Aula 13 - Exemplo 5
 * Arquivo commissionCap13Ex5.cpp
 * Autor: Miguel Campista
 */
#include "commissionCap13Ex5.h"

CommissionEmployee::CommissionEmployee (const string &first, const string &last,
 const string &ssn, double sales, double rate) :
 Employee (first, last, ssn) {
 setGrossSales (sales);
 setCommissionRate (rate);
}

void CommissionEmployee::setCommissionRate (double rate) {
 commissionRate = (rate > 0.0 && rate < 1.0) ? rate : 0.0;
}

double CommissionEmployee::getCommissionRate () const { return commissionRate; }

void CommissionEmployee::setGrossSales (double sales) {
 grossSales = (sales < 0.0) ? 0.0 : sales;
}

double CommissionEmployee::getGrossSales () const { return grossSales; }

// Calcula os rendimentos
// Sobrescreve a fç virtual para earnings em Employee
double CommissionEmployee::earnings () const {
 return getGrossSales () * getCommissionRate ();
}

void CommissionEmployee::print () const {
 cout << "Commission employee: ";
 Employee::print ();
 cout << "\nGross sales: " << getGrossSales ()
 << "\nCommission rate: " << getCommissionRate ();
}

```

## Cl. Derivada Concreta Indireta BasePlusCommissionEmployee

- **BasePlusCommissionEmployee herda de CommissionEmployee**
  - Inclui o salário-base
    - A função **earnings** sobrescrita incorpora o salário-base
    - A função **print** sobrescrita incorpora o salário-base
  - Classe concreta, porque a classe derivada é concreta
    - Não é necessário sobrescrever **earnings** para torná-la concreta. É possível herdar a implementação de **CommissionEmployee**
      - Embora sobrescrevamos **earnings** para incorporar o **salário-base**

POO para Redes de Computadores - COPPE-PEE/UFRJ

Prof. Miguel Campista

## Quinto Exemplo de Polimorfismo em C++

```

/*
 * Aula 13 - Exemplo 5
 * Arquivo basepluscommissionCap13Ex5.h
 * Autor: Miguel Campista
 */
#ifndef BASEPLUS_H
#define BASEPLUS_H

#include <iostream>
#include <string>
#include <iomanip>
#include "commissionCap13Ex5.h"

using namespace std;

class BasePlusCommissionEmployee : public CommissionEmployee {
public:
 BasePlusCommissionEmployee (const string &, const string &,
 const string &, double = 0.0, double = 0.0, double = 0.0);

 void setBaseSalary (double); // Conf. o salário base
 double getBaseSalary () const;

 // Palavra-chave virtual assinala a intenção de sobrescrever
 virtual double earnings () const; // Calcula os rendimentos
 virtual void print () const; // Imprime os objetos HourlyEmployee

private:
 double baseSalary;
};

#endif

```

## Quinto Exemplo de Polimorfismo em C++

```

/*
 * Aula 13 - Exemplo 5
 * Arquivo basepluscommissionCap13Ex5.cpp
 * Autor: Miguel Campista
 */
#include "basepluscommissionCap13Ex5.h"

BasePlusCommissionEmployee::BasePlusCommissionEmployee (const string &first,
 const string &last, const string &ssn,
 double sales, double rate, double salary) :
 CommissionEmployee (first, last, ssn, sales, rate) {
 setBaseSalary (salary);
}

void BasePlusCommissionEmployee::setBaseSalary (double salary) {
 baseSalary = (salary < 0.0) ? 0.0 : salary;
}

double BasePlusCommissionEmployee::getBaseSalary () const { return baseSalary; }

double BasePlusCommissionEmployee::earnings () const {
 return getBaseSalary () + CommissionEmployee::earnings ();
}

void BasePlusCommissionEmployee::print () const {
 cout << "base salary: ";
 CommissionEmployee::print (); // reuse o código
 cout << "\nbase salary: " << getBaseSalary ();
}

```

# Demonstração do Processo Polimórfico

- Crie objetos do tipo **SalariedEmployee**, **HourlyEmployee**, **CommissionEmployee** e **BasePlusCommissionEmployee**
  - Demonstre a manipulação de objetos com a vinculação estática
    - Usando **handles de nome em vez de ponteiros ou referências**
    - O compilador pode identificar cada tipo de objeto para determinar que função **print** e **earnings** deve chamar
  - Demonstre a manipulação de objetos polimorficamente
    - Use um **vetor de ponteiros Employee**
    - Invoque funções **virtual** usando **ponteiros e referências**

POO para Redes de Computadores - COPPE-PEE/UFRJ Prof. Miguel Campista

```

/*
 * Aula 13 - Exemplo 8
 * Programa Principal
 * Autor: Miguel Campista
 */
#include <iostream>
#include <iomanip>
#include <vector>

#include "employeeCap13Ex8.h"
#include "salariedEmployeeCap13Ex8.h"
#include "hourlyEmployeeCap13Ex8.h"
#include "commissionCap13Ex8.h"
#include "basePlusCommissionCap13Ex8.h"

void virtualViaPointer (const Employee * const); // Protótipo
void virtualViaReference (const Employee &); // Protótipo

int main () {
 // formatação da saída do ponto flutuante
 cout << fixed << setprecision(2);

 // cria objetos da classe derivada
 SalariedEmployee salariedEmployee("John", "Smith", "111-11-1111", 800);
 HourlyEmployee hourlyEmployee(
 "Karen", "Price", "222-22-2222", 16.75, 40);
 CommissionEmployee commissionEmployee(
 "Bob", "Jones", "333-33-3333", 10000, .06);
 BasePlusCommissionEmployee basePlusCommissionEmployee(
 "Bob", "Lewis", "444-44-4444", 5000, .04, 300);

 cout << "Employees processed individually using static binding:\n\n";
}

```

# Quinto Exemplo de Polimorfismo em C++

```

// chama funções virtuais de Employee virtual print e earnings de um
// ponteiro para classe base usando binding dinâmico
void virtualViaPointer (const Employee * const baseClassPtr) {
 baseClassPtr->print();
 cout << "\nearnings $ " << baseClassPtr->earnings() << "\n\n";
}

// chama funções virtuais de Employee virtual print e earnings de um
// referência para classe base usando binding dinâmico
void virtualViaReference (const Employee &baseClassRef) {
 baseClassRef.print();
 cout << "\nearnings $ " << baseClassRef.earnings() << "\n\n";
}

```

POO para Redes de Computadores - COPPE-PEE/UFRJ Prof. Miguel Campista

```

// imprime a informação de cada Employee e
// rendimentos com binding estático
salariedEmployee.print();
hourlyEmployee.print();
commissionEmployee.print();
basePlusCommissionEmployee.print();
cout << "\nearnings $ " << salariedEmployee.earnings() << "\n\n";
cout << "\nearnings $ " << hourlyEmployee.earnings() << "\n\n";
cout << "\nearnings $ " << commissionEmployee.earnings() << "\n\n";
cout << "\nearnings $ " << basePlusCommissionEmployee.earnings()
 << "\n\n";

// cria vetor de ponteiros de quatro classe bases
vector <Employee * > employees (4);

// inicializa vetor com Employees
employees [0] = &salariedEmployee;
employees [1] = &hourlyEmployee;
employees [2] = &commissionEmployee;
employees [3] = &basePlusCommissionEmployee;

cout << "Employees processed polymorphically via dynamic binding:\n\n";

// chama virtualViaPointer para imprimir informação de cada Employee
// e rendimentos usando binding dinâmico
cout << "Virtual function calls made off base-class pointers:\n\n";
for (size_t i = 0; i < employees.size(); i++)
 virtualViaPointer (employees [i]);

// chama virtualViaReference para imprimir informação de cada Employee
// e rendimentos usando binding dinâmico
cout << "Virtual function calls made off base-class references:\n\n";
for (size_t i = 0; i < employees.size(); i++)
 virtualViaReference (*employees [i]); // note dereferencing

return 0;
}

```

# Quinto Exemplo de Polimorfismo em C++

```

// chama funções virtuais de Employee virtual print e earnings de um
// ponteiro para classe base usando binding dinâmico
void virtualViaPointer (const Employee * const baseClassPtr) {
 baseClassPtr->print();
 cout << "\nearnings $ " << baseClassPtr->earnings() << "\n\n";
}

// chama funções virtuais de Employee virtual print e earnings de um
// referência para classe base usando binding dinâmico
void virtualViaReference (const Employee &baseClassRef) {
 baseClassRef.print();
 cout << "\nearnings $ " << baseClassRef.earnings() << "\n\n";
}

```

# Quinto Exemplo de Polimorfismo em C++

```

// chama funções virtuais de Employee virtual print e earnings de um
// ponteiro para classe base usando binding dinâmico
void virtualViaPointer (const Employee * const baseClassPtr) {
 baseClassPtr->print();
 cout << "\nearnings $ " << baseClassPtr->earnings() << "\n\n";
}

// chama funções virtuais de Employee virtual print e earnings de um
// referência para classe base usando binding dinâmico
void virtualViaReference (const Employee &baseClassRef) {
 baseClassRef.print();
 cout << "\nearnings $ " << baseClassRef.earnings() << "\n\n";
}

```

POO para Redes de Computadores - COPPE-PEE/UFRJ Prof. Miguel Campista

## Estudo de Caso: Sistema de Folha de Pagamento

- Sistema de Folha de Pagamento utilizando polimorfismo e informações de tipo em tempo de execução com downcasting, `dynamic_cast`, `typeid` e `type_info`
  - Ex.: Recompense `BasePlusCommissionEmployee` adicionando 10% a seus salários-base
- É necessário usar informações de tipo em tempo de execução e coerção dinâmica para "programar no específico"
  - Alguns compiladores exigem que essas informações sejam habilitadas para serem utilizadas no programa
    - Consulte a documentação do compilador

POO para Redes de Computadores - COPPE-PEE/UFRJ

Prof. Miguel Campista

## Estudo de Caso: Sistema de Folha de Pagamento

- Operador `dynamic_cast`
  - Operação downcast
    - Converte um ponteiro de classe base em um ponteiro de classe derivada
  - Se um objeto subjacente for do tipo derivado, será executada a coerção
    - Do contrário, será atribuído 0
  - Se `dynamic_cast` não for utilizado e houver a tentativa de atribuir um ponteiro de classe base a um ponteiro de classe derivada
    - Ocorrerá um erro de compilação

POO para Redes de Computadores - COPPE-PEE/UFRJ

Prof. Miguel Campista

## Estudo de Caso: Sistema de Folha de Pagamento

- Operador `typeid`
  - Recebe um ponteiro desreferenciado e
  - Retorna uma referência a um objeto da classe `type_info`
    - Contém informações sobre o tipo de seu operando
    - Função-membro `name ()`
      - Retorna uma string baseada em ponteiro que contém o nome do tipo do argumento passado para `typeid`
  - Deve incluir o arquivo de cabeçalho `<typeinfo>`

POO para Redes de Computadores - COPPE-PEE/UFRJ

Prof. Miguel Campista

```

/*
 * Aula 13 - Exemplo 6
 * Programa Principal
 * Autor: Miguel Campista
 */
#include <iostream>
#include <iomanip>
#include <vector>
#include <typeinfo>

#include "employeeCap13Ex6.h"
#include "salariedEmployeeCap13Ex6.h"
#include "hourlyEmployeeCap13Ex6.h"
#include "commissionCap13Ex6.h"
#include "basePlusCommissionCap13Ex6.h"

int main () {
 // formatação de saída do ponto flutuante
 cout << fixed << setprecision(2);

 // cria vetor de ponteiros de quatro classe bases
 vector <Employee *> employees (4);

 // cria objetos da classe derivada
 employees [0] = new SalariedEmployee("John", "Smith", "111-11-1111", 800);

 employees [1] = new HourlyEmployee(
 "Karen", "Price", "222-22-2222", 16.75, 40);

 employees [2] = new CommissionEmployee(
 "Sue", "Jones", "333-33-3333", 10000, .06);

 employees [3] = new BasePlusCommissionEmployee(
 "Bob", "Lewis", "444-44-4444", 8000, .04, 300);

 employees [i]->print ();
}

```

## Sexto Exemplo de Polimorfismo em C++

```

// Processa polimorficamente cada elemento no vetor employees
for (size_t i = 0; i < employees.size(); i++) {
 employee& e = *employees[i];
 cout << endl;

 // Ponteiro downcast
 BasePlusCommissionEmployee *derivedPtr =
 dynamic_cast <BasePlusCommissionEmployee *> (employees [i]);

 // determina se o elemento aponta para o empregado comissionado
 // com salário base
 if (derivedPtr != 0) {
 double oldBaseSalary = derivedPtr->getBaseSalary ();
 cout << "old base salary: $" << oldBaseSalary << endl;
 derivedPtr->setBaseSalary (1.10 * oldBaseSalary);
 cout << "new base salary with 10% increase is: $"
 << derivedPtr->getBaseSalary () << endl;
 }

 cout << "earned $" << employees [i]->earnings () << "\n\n";
}

// libera objetos apontando pelos elementos do vetor
for (size_t i = 0; i < employees.size(); i++) {
 cout << "deleting object of "
 << typeid(*employees [i]).name () << endl;
 delete employees [i];
}

return 0;
}

```

## Sexto Exemplo de Polimorfismo em C++

```

// Processa polimorficamente cada elemento no vetor employees
for (size_t i = 0; i < employees.size(); i++) {
 employee& e = *employees[i];
 cout << endl;

 // Ponteiro downcast
 BasePlusCommissionEmployee *derivedPtr =
 dynamic_cast <BasePlusCommissionEmployee *> (employees [i]);

 // determina se o elemento aponta para o empregado comissionado
 // com salário base
 if (derivedPtr != 0) {
 double oldBaseSalary = derivedPtr->getBaseSalary ();
 cout << "old base salary: $" << oldBaseSalary << endl;
 derivedPtr->setBaseSalary (1.10 * oldBaseSalary);
 cout << "new base salary with 10% increase is: $"
 << derivedPtr->getBaseSalary () << endl;
 }

 cout << "earned $" << employees [i]->earnings () << "\n\n";
}

// libera objetos apontando pelos elementos do vetor
for (size_t i = 0; i < employees.size(); i++) {
 cout << "deleting object of "
 << typeid(*employees [i]).name () << endl;
 delete employees [i];
}

return 0;
}

```

```

Salaried employee: John Smith
social security number: 111-11-1111
newly salary: 800.00
earned 3800.00

hourly employee: Karen Price
social security number: 222-22-2222
gross wage: 16.75; hours worked: 40.00
earned 2670.00

commission employee: Sue Jones
social security number: 333-33-3333
gross sale: 10000.00
commission rate: 0.06
earned 5600.00

base salary commission employee: Bob Lewis
social security number: 444-44-4444
gross sale: 8000.00
commission rate: 0.04
base salary: 300.00
old base salary: 300.00
new base salary with 10% increase is: 330.00
earned 5330.00

deleting object of 16SalariedEmployee
deleting object of 18HourlyEmployee
deleting object of 19CommissionEmployee
deleting object of 21BasePlusCommissionEmployee
Pressione qualquer tecla para continuar.

```

## Destrutores Virtuais

- Destrutores não virtuais
  - Destrutores que não são declarados com a palavra-chave `virtual`
  - Se um objeto de classe derivada for destruído explicitamente atribuindo o operador `delete` a um ponteiro de classe base para o objeto, o comportamento será indefinido

POO para Redes de Computadores - COPPE-PEE/UFRJ

Prof. Miguel Campista

## Destrutores Virtuais

- Destrutores **virtual**
  - São declarados com a palavra-chave `virtual`
    - Todos os destrutores de classe base são **virtual**
  - Se um objeto de classe derivada for destruído explicitamente aplicando o operador `delete` a um ponteiro de classe base para um objeto, o destrutor de classe derivada apropriado será chamado
    - Destrutores de classe base apropriados **executarão posteriormente**

POO para Redes de Computadores - COPPE-PEE/UFRJ

Prof. Miguel Campista

## Destrutores Virtuais

- Se uma classe tiver funções **virtual**, forneça um destrutor **virtual**, mesmo que ele não seja requerido para a classe
  - As classes derivadas dessa classe podem conter destrutores que devem ser chamados adequadamente
- Os construtores não podem ser **virtual**
  - Declarar um construtor **virtual** é um erro de compilação

POO para Redes de Computadores - COPPE-PEE/UFRJ

Prof. Miguel Campista

## Exemplo 2

- Escreva um programa que implemente a classe **EmployeeCadastro** que herda atributos e métodos das classes **Cadastro** e **Senha**. Implemente também a classe **ClientCadastro** que herda somente atributos e métodos da classe **Cadastro**. Todas as classes devem implementar um método **virtual print** que para imprimir os valores de todos os seus atributos. A função principal deve instanciar dinamicamente ponteiros da classe base **Cadastro** para objetos de cada uma das classes derivadas.



POO para Redes de Computadores - COPPE-PEE/UFRJ

Prof. Miguel Campista

## Exemplo 2

```
/*
 * Aula 13 - Exemplo 2
 * Arquivo cadastroCap13Ex7.h
 * Autor: Miguel Campista
 */
#ifndef CADASTRO_H
#define CADASTRO_H

#include <iostream>
#include <string>

using namespace std;

class Cadastro {
public:
 Cadastro (string, int);
 virtual ~Cadastro ();

 string getName () const;
 int getAge () const;
 virtual void print () const;

private:
 string name;
 int age;
};

#endif
```

POO para Redes de Computadores - COPPE-PEE/UFRJ

Prof. Miguel Campista

## Exemplo 2

```
/*
 * Aula 13 - Exemplo 2
 * Arquivo cadastroCap13Ex7.cpp
 * Autor: Miguel Campista
 */
#include "cadastroCap13Ex7.h"

Cadastro::Cadastro (string n, int a) : name (n), age (a) {}

Cadastro::~Cadastro () {
 cout << "Destrutor da classe Cadastro..." << endl;
}

string Cadastro::getName () const { return name; }

int Cadastro::getAge () const { return age; }

void Cadastro::print () const {
 cout << "Name: " << getName ()
 << "\nAge: " << getAge () << endl;
}
```

POO para Redes de Computadores - COPPE-PEE/UFRJ

Prof. Miguel Campista



## Exemplo 2

```
/*
 * Aula 13 - Exemplo 7
 * Arquivo senhaCap13Ex7.h
 * Autor: Miguel Campista
 */
#ifndef SENHA_H
#define SENHA_H

#include <iostream>
#include <string>

using namespace std;

class Senha {
public:
 Senha (string);
 virtual ~Senha ();

 string getSenha () const;
 virtual void print () const;

private:
 string senha;
};

#endif
```

POO para Redes de Computadores - COPPE-PEE/UFRJ

Prof. Miguel Campista

## Exemplo 2

```
/*
 * Aula 13 - Exemplo 7
 * Arquivo senhaCap13Ex7.cpp
 * Autor: Miguel Campista
 */
#include "senhaCap13Ex7.h"

Senha::Senha (string s) : senha (s) {}

Senha::~Senha () {
 cout << "Destrutor da classe Senha..." << endl;
}

string Senha::getSenha () const { return senha; }

void Senha::print () const {
 cout << "Senha: " << getSenha () << endl;
}
```

POO para Redes de Computadores - COPPE-PEE/UFRJ

Prof. Miguel Campista

## Exemplo 2

```
/*
 * Aula 12 - Exemplo 7
 * Arquivo employeecadastroCap12Ex7.h
 * Autor: Miguel Campista
 */
#ifndef EMPLOYEECADASTRO_H
#define EMPLOYEECADASTRO_H

#include <iostream>
#include <string>
#include "cadastroCap13Ex7.h"
#include "senhaCap13Ex7.h"

using namespace std;

class EmployeeCadastro : public Cadastro, public Senha {
public:
 EmployeeCadastro (string, int, string, string);
 virtual ~EmployeeCadastro ();

 string getJob () const;
 virtual void print () const;

private:
 string job;
};

#endif
```

POO para Redes de Computadores - COPPE-PEE/UFRJ

Prof. Miguel Campista

## Exemplo 2

```
/*
 * Aula 12 - Exemplo 7
 * Arquivo employeecadastroCap12Ex7.cpp
 * Autor: Miguel Campista
 */
#include "employeecadastroCap12Ex7.h"

EmployeeCadastro::EmployeeCadastro (string n, int a, string s, string j)
 : Cadastro (n, a), Senha (s), job (j) {}

EmployeeCadastro::~EmployeeCadastro () {
 cout << "Destrutor da classe EmployeeCadastro..." << endl;
}

string EmployeeCadastro::getJob () const { return job; }

void EmployeeCadastro::print () const {
 Cadastro::print ();
 Senha::print ();
 cout << "Job: " << getJob () << "\n\n" << endl;
}
```

POO para Redes de Computadores - COPPE-PEE/UFRJ

Prof. Miguel Campista

## Exemplo 2

```
/*
 * Aula 13 - Exemplo 7
 * Arquivo clientcadastroCap13Ex7.h
 * Autor: Miguel Campista
 */
#ifndef CLIENTCADASTRO_H
#define CLIENTCADASTRO_H

#include <iostream>
#include <string>
#include "cadastroCap13Ex7.h"

using namespace std;

class ClientCadastro : public Cadastro {
public:
 ClientCadastro (string, int, string);
 virtual ~ClientCadastro ();

 string getPreference () const;
 virtual void print () const;

private:
 string preference;
};

#endif
```

POO para Redes de Computadores - COPPE-PEE/UFRJ

Prof. Miguel Campista

## Exemplo 2

```
/*
 * Aula 13 - Exemplo 7
 * Arquivo clientcadastroCap13Ex7.cpp
 * Autor: Miguel Campista
 */
#include "clientcadastroCap13Ex7.h"

ClientCadastro::ClientCadastro (string n, int a, string p)
 : Cadastro (n, a), preference (p) {}

ClientCadastro::~ClientCadastro () {
 cout << "Destrutor da classe ClientCadastro..." << endl;
}

string ClientCadastro::getPreference () const { return preference; }

void ClientCadastro::print () const {
 Cadastro::print ();
 cout << "Preference: " << getPreference () << "\n\n" << endl;
}
```

POO para Redes de Computadores - COPPE-PEE/UFRJ

Prof. Miguel Campista

## Exemplo 2

```
/*
 * Aula 13 - Exemplo 7
 * Programa Principal
 * Autor: Miguel Campista
 */
#include "EmployeeCadastroCap13Ex7.h"
#include "ClientCadastroCap13Ex7.h"

int main () {
 Cadastro cad (2);

 cad [0] = new EmployeeCadastro ("Joao das Covas", 30, "abc", "Engenheiro");
 cad [1] = new ClientCadastro ("Jose Mineiro", 20, "Esportes");

 cad [0]->print ();
 cout << endl;
 cad [1]->print ();

 delete cad [0];
 delete cad [1];

 return 0;
}
```

## Exemplo 2

```
/*
 * Aula 13 - Exemplo 7
 * Programa Principal
 * Autor: Miguel Campista
 */
C:\Users\Miguel\Documents\UFRJ\disciplina\linguagem\projeto\aula13-ex7.exe
Nome: Joao das Covas
Age: 30
Email: abc
Job: Engenheiro

Nome: Jose Mineiro
Age: 20
Preference: Esportes

Destrutor da classe EmployeeCadastro...
Destrutor da classe Emha...
Destrutor da classe Cadastro...
Destrutor da classe ClientCadastro...
Destrutor da classe Cadastro...
Pressione qualquer tecla para continuar. . .
```

## Leitura Recomendada

- Capítulos 13 do livro
  - Deitel, "C++ How to Program", 5th edition, Editora Prentice Hall, 2005