

Aula 6

HERANÇA E POLIMORFISMO

PROF. SANDINO JARDIM

CC-UFMT-CUA

Idéias-chave - POO

- Aula 4...

POO – Visão Geral

- Idéias-chave:
 - Abstração de dados
 - Separação entre interface e implementação
 - Herança
 - Modelagem de relacionamentos entre tipos similares
 - Vinculação dinâmica
 - Uso de objetos similares, ignorando detalhes que os diferem

Motivação

Considere o seguinte problema:

- Em uma empresa é necessário guardar os seguintes dados sobre um empregado: nome, salário e CPF.
- Também é preciso fornecer métodos nos quais tais valores possam ser retornados;
- Anualmente, os empregados recebem um percentual de aumento segundo índice de inflação;
- Diferentemente de outros empregados, os Gerentes dessa empresa recebem o salário mensal acrescido de uma bonificação

Como modelar esse problema?

Motivação

Primeira solução - modelar cada classe individualmente

Empregado
<ul style="list-style-type: none">- nome: String- salario: double- cpf :int
<ul style="list-style-type: none">+ getName() : String+ getSalario() : double+ getCPF : int+ aumentaSalario(percentual:double): void

Gerente
<ul style="list-style-type: none">- nome: String- salario: double- cpf : int- bonus : double
<ul style="list-style-type: none">+ getName() : String+ getSalario() : double+ getCPF : int+ aumentaSalario(percentual:double): void+ getBonus() : double

Motivação

Primeira solução – problema

- Código duplicado!
- Se a regra de negócio que rege esse comportamento mudar, teremos que mudar o código em duas classes

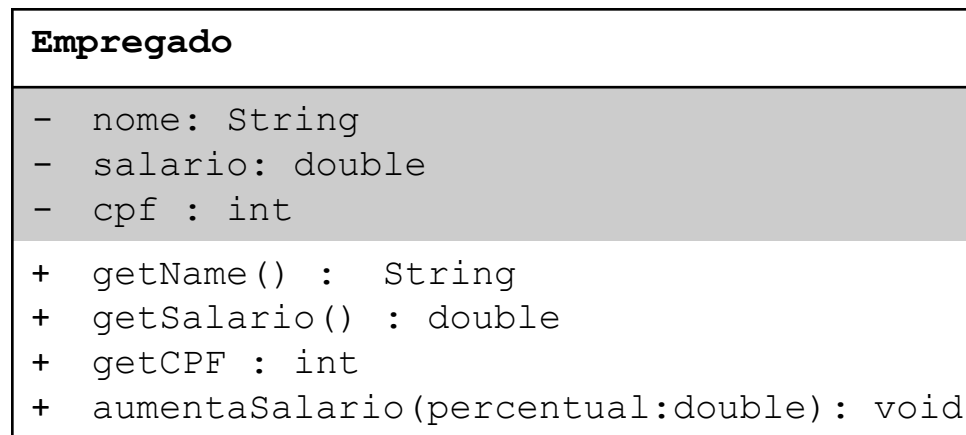
Motivação

Segunda solução

- Definir a classe Gerente como uma **especialização** da classe Empregado
- Gerente **é um** Empregado, portanto, Gerente é uma subcategoria de Empregado

Motivação

Segunda solução



Implementação de
Empregado não se altera

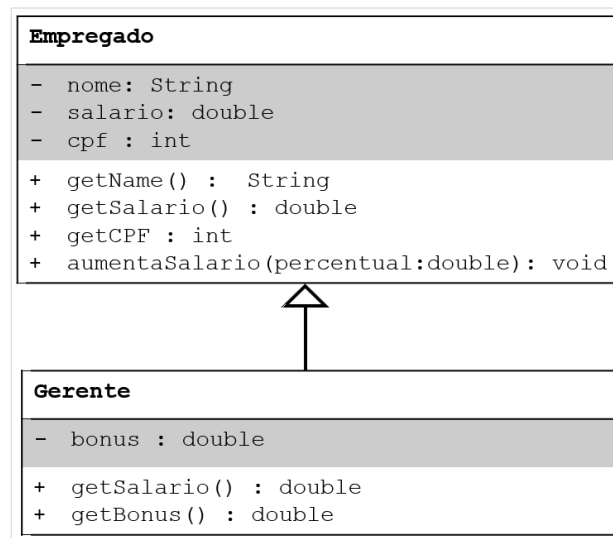
Gerente passa a ser uma
subclasse da classe
Empregado

Herança

Permite criar novas classes a partir de classes existentes

A classe que herda (subclasse) reutiliza os atributos e métodos da classe herdada (superclasse)

A subclasse (classe derivada) pode adicionar novos atributos ou métodos, além de sobrescrever métodos da superclasse (classe base)



Herança em C++

Classe Base: Empregado

```
class Empregado{
private:
    string nome;
    int cpf;
    double salario;
public:
    Empregado(string nome, double salario, int cpf):
        nome(nome), salario(salario), cpf(cpf){}
    string getNome(){return nome;}
    double getSalario(){return salario;}
    void aumentaSalario(double percent){salario*=percent;}
    //virtual ~Empregado() = default;
};
```

Herança em C++

Classe Derivada: Gerente

```
class Gerente : public Empregado{
private:
    double bonus;
public:
    Gerente(string nome, double salario, int cpf, double bonus):
        Empregado(nome, salario, cpf),
        bonus(bonus) {}

    double getBonus() {return bonus;}
};
```

Herança em C++

Aplicação

```
int main() {  
    Empregado e("Fulano", 2500.00, 123);  
    Gerente g("Beltrano", 3500.00, 456, 500.00);  
  
    cout << e.getSalario();  
    cout << g.getSalario() + g.getBonus();  
}
```

Saída:

2500.00

4000.00

Herança em C++

Solução 1 – operador de escopo

```
class Gerente : public Empregado{
private:
    double bonus;
public:
    Gerente(string nome, double salario, int cpf, double bonus):
        Empregado(nome, salario, cpf),
        bonus(bonus) {}
    double getSalario() {return Empregado::getSalario() + bonus;}
    double getBonus() {return bonus;}
};
```

Herança em C++

Solução 2: visibilidade `protected`

- Membros deste nível podem ser acessados diretamente também pelas subclasses

Classe Base: Empregado (3)

```
class Empregado{
private:
    string nome;
    int cpf;
protected:
    double salario;
public:
    Empregado(string nome, double salario, int cpf):
        nome(nome), salario(salario), cpf(cpf){}
    string getNome(){return nome;}
    virtual double getSalario(){return salario;}
    void aumentaSalario(double percent){salario*=percent;}
    //virtual ~Empregado() = default;
};
```

Herança em C++

Classe Derivada: Gerente (2 [agora possível após solução 2])

```
class Gerente : public Empregado{
private:
    double bonus;
public:
    Gerente(string nome, double salario, int cpf, double bonus):
        Empregado(nome, salario, cpf),
        bonus(bonus) {}
    double getSalario() override {return salario+bonus;}
    double getBonus() {return bonus;}
};
```

Herança em C++

Aplicação (2)

```
int main() {  
    Empregado e("Fulano", 2500.00, 123);  
    Gerente g("Beltrano", 3500.00, 456, 500.00);  
  
    cout << e.getSalario() << endl;  
    cout << g.getSalario() << endl;  
}
```

Saída:

2500.00

4000.00

Polimorfismo

Capacidade de um objeto se comportar de diferentes maneiras.

Gerar um resultado diferente ao ser invocado um mesmo método

Em C++ é alcançável pelo uso de ponteiros

```
int main() {  
    Empregado e("Fulano", 2500.00, 123);  
    Gerente g("Beltrano", 3500.00, 456, 500.00);  
  
    Empregado *e2 = &e;  
    cout << e2->getSalario() << endl;  
    e2 = &g;  
    cout << e2->getSalario() << endl;  
}
```


Polimorfismo

Condições:

1. Ponteiros genéricos
2. Objetos intercambiáveis
3. Funções virtuais

Exemplo de uso:

- Considerando que os empregados podem ser de dois tipos, implementar uma aplicação que calcule a folha salarial considerando todas as categorias de empregados.

Herança em C++

Classe Base: Empregado (2)

```
class Empregado{
private:
    string nome;
    int cpf;
    double salario;
public:
    Empregado(string nome, double salario, int cpf):
        nome(nome), salario(salario), cpf(cpf){}
    string getNome(){return nome;}
    virtual double getSalario(){return salario;}
    void aumentaSalario(double percent){salario*=percent;}
    //virtual ~Empregado() = default;
};
```

Indica às classes derivadas a intenção de reescrever o método

Herança em C++

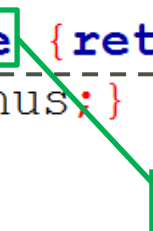
Classe Derivada: Gerente (2)

```
class Gerente : public Empregado{  
private:  
    double bonus;  
public:  
    Gerente(string nome, double salario, int cpf, double bonus) :  
        Empregado(nome, salario, cpf),  
        bonus(bonus) {}  
    double getSalario() override {return salario+bonus;}  
    double getBonus() {return bonus;}  
};
```

Problema: acesso **private**



Palavra-chave indicando
sobrescrita do método



Herança em C++

Classe Derivada: Gerente (3)

```
class Gerente : public Empregado{
private:
    double bonus;
public:
    Gerente(string nome, double salario, int cpf, double bonus):
        Empregado(nome, salario, cpf),
        bonus(bonus) {}
    double getSalario() override {return getSalario()+bonus;}
    double getBonus() {return bonus;}
};
```

Novo problema: redundância



Solução sem ponteiro

```
int main() {
    Empregado e1("Fulano", 2500.00, 123);
    Empregado e2("Giuliano", 2700.00, 888);
    Empregado e3("Damiano", 3580.00, 999);
    Gerente g1("Beltrano", 3500.00, 456, 500.00);
    Gerente g2("Siclano", 8500.00, 777, 900.00);
    double totalFolha = 0.0;
    vector<Empregado> empregados;
    empregados.push_back(e1);
    empregados.push_back(e2);
    empregados.push_back(e3);

    vector<Gerente> gerentes;
    gerentes.push_back(g1);
    gerentes.push_back(g2);

    for(auto e : empregados) {
        totalFolha += e.getSalario();
    }
    for(auto g : gerentes) {
        totalFolha += g.getSalario() << endl;
    }
}
```

Solução completa poliformismo

```
int main() {
    Empregado e1("Fulano", 2500.00, 123);
    Empregado e2("Giuliano", 2700.00, 888);
    Empregado e3("Damiano", 3580.00, 999);
    Gerente g1("Beltrano", 3500.00, 456, 500.00);
    Gerente g2("Siclano", 8500.00, 777, 900.00);
    double totalFolha = 0.0;
    vector<Empregado*> funcionarios;
    funcionarios.push_back(&e1);
    funcionarios.push_back(&e2);
    funcionarios.push_back(&e3);
    funcionarios.push_back(&g1);
    funcionarios.push_back(&g2);

    for(auto f : funcionarios){
        totalFolha += f->getSalario();
    }
}
```